This chapter describes how to expand the capabilities of the AOCE Catalogs Extension to the Finder. The AOCE Catalogs Extension (CE) allows users to use the Finder's iconographic interface to search through AOCE catalogs, examine records, and edit records. You can provide extensions to the CE that make it possible for the Finder to handle new types of records and attributes, to group record types in a new way, or to present the content of records and record attributes in a new way. These extensions to the AOCE Catalogs Extension are called ***AOCE templates***.

This chapter describes the various types of AOCE templates, tells you how to write an AOCE template, and defines the resource types that you use to create an AOCE template. How you apply AOCE templates is up to you. Indeed, the entire point of AOCE templates is to allow developers to extend the CE in ways that could not be foreseen.

This chapter is intended for developers who are providing new record types or new attribute types and who want to allow users to use the AOCE Catalogs Extension to the Finder to view, create, or modify these records and attributes. The chapter is also for anyone who wants to extend the capabilities of the CE in any other way. This chapter is also required reading along with the chapter "Service Access Module Setup" in *Inside Macintosh: AOCE Service Access Modules* for anyone writing a service access module (SAM).

To use this chapter, you should be familiar with the structure of AOCE catalogs as described in the introduction to the chapter "Catalog Manager" in this book. You also must be familiar with Macintosh resources as described in the chapter on the Resource Manager in *Inside Macintosh: More Macintosh Toolbox*. The sample code listings in this chapter include code written for the Rez resource compiler.

This chapter first briefly describes the human interface of the AOCE Catalogs Extension and the way in which AOCE templates work together to modify the CE. Next it describes the use of each type of AOCE template in more detail. Finally, the chapter presents the details of the implementation of AOCE templates, including definitions of the various resource types you use to create AOCE templates.

Introduction to the AOCE Catalogs Extension

AOCE catalogs, described in the chapter "Catalog Manager" in this book, contain information arranged in a hierarchical structure similar to the Macintosh hierarchical file system (HFS). At the root level of the hierarchy is the AOCE catalog itself. Each catalog can contain any number of dNodes, and each dNode can contain dNodes and records, which contain the data.

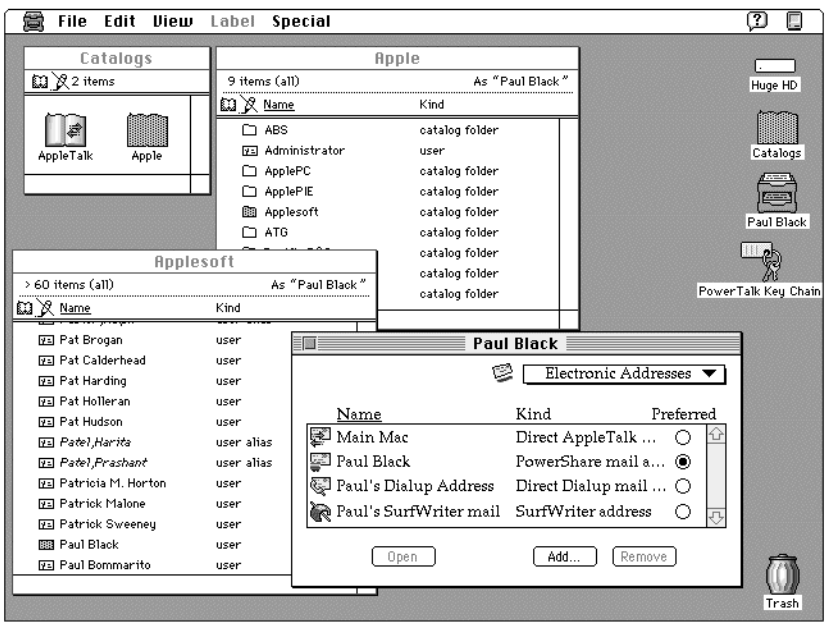
The Catalog Manager provides access to server-based AOCE catalogs, including those implemented by Apple Computer, Inc.'s PowerShare servers and those implemented by third parties through the CSAM interface. The Catalog Manager also provides access to catalogs on the local Macintosh computer: personal catalog files and information cards (individual records on disk).

Catalogs are organized in a hierarchy of dNodes, much like Macintosh HFS folders. Each dNode has a name, so that a particular location within the catalog can be specified by listing the dNode names leading from the catalog through the hierarchy to the particular dNode of interest. At the bottom of the hierarchy are records, just as Macintosh files are at the bottom of the HFS hierarchy. Each record has a name and a type. The data within records is organized into attributes, each of which can contain an arbitrary number of values. Each attribute has a type, and each attribute value has a *tag* that indicates its format. Attribute tags are of type `OSType` and are therefore 4 bytes long. Attribute values are blocks of data up to 64 KB in size.

PowerShare catalogs and the CE also support stand-alone attributes. A ***stand-alone attribute*** is a record that contains only one attribute, extracted from another record. Although technically a record, the AOCE software treats a stand-alone attribute like an attribute in most circumstances. For example, if a user drags a single mail address from a User record and drops it in a personal catalog, the CE creates a stand-alone attribute containing that address. When the user drops that stand-alone attribute onto another User record, the AOCE software adds the address to the User record as an attribute.

The AOCE Catalogs Extension to the Finder places all PowerShare catalogs and CSAM catalogs within an icon (the "Catalogs" icon) that must remain on the desktop like the icon for a disk. To the user, each catalog within the Catalogs icon appears to be a folder, and each dNode within the catalog appears to be a folder inside another folder. Figure 5-1 shows what a user's desktop might look like with catalogs and dNodes open.

Figure 5-1 The AOCE Catalogs Extension in use



From the user's perspective, browsing the catalog system is very much like browsing the Macintosh file system. Catalog dNodes look like file folders. They contain lists of enclosed dNodes and records. Opening a dNode produces a new window showing the contents of that dNode.

There are three major differences between browsing HFS directories and browsing AOCE catalogs:

- The AOCE Catalogs Extension can display the contents of catalogs that contain many more items than could be browsed in the file system. While file system browsing is limited to hundreds of items, catalog dNodes can contain tens of thousands of items.
- Finder windows that display lists of AOCE catalogs, contents of catalogs, and contents of dNodes include a column labeled "Kind," just as Finder windows for HFS folders include a "Kind" column. Whereas HFS windows differentiate such items as application documents, application programs, and folders, AOCE windows differentiate between PowerShare catalogs and CSAM catalogs, and among records of various types. Unlike the file system, AOCE catalog items are grouped into categories such as people, mail servers, and AppleShare. For example, the separate record types for LaserWriters, ImageWriters, and ImageWriter LCs could be grouped into the category "printers." The user can use the View menu to select the categories to be displayed (Figure 5-2).
- Whereas the Finder launches an application to display the contents of HFS files, the AOCE Catalogs Extension itself can display and be used to edit the contents of records.

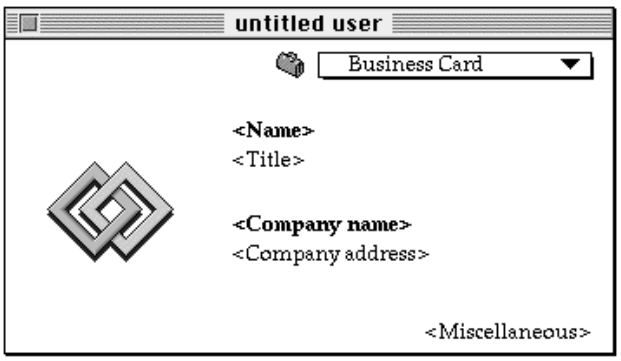
Figure 5-2 View menu seen with the AOCE Catalogs Extension to the Finder

When the user opens a record, the AOCE Catalogs Extension opens a window that lets the user select any of a series of information pages. Each ***information page*** (sometimes called an *info-page*) shows a portion of the contents of the record. The user moves to different information pages by using a pop-up menu.

Depending on the type of record and the design of the information pages, the user might be able to select various options such as different displays of the same data, perform functions such as dialing the telephone, and edit certain fields of the information page or

even create a new record. Figure 5-3 shows an information page for an address catalog. Notice the pop-up menu and editable text fields.

Figure 5-3 Information page



Any information page for a record can include a list of attribute values. The information page template specifies which attribute types are included in the list. Because the list includes a subset of the attribute types in the record and appears as a distinct portion of the information page window, these lists are referred to as ***sublists***. Each sublist entry includes information from within the attribute value as specified by the template. After opening an item in a sublist, the user is presented with a new information page window displaying the contents of the opened attribute. Similarly, a dNode window can include a sublist of records contained in the dNode. Figure 5-4 shows an information page with a sublist, and Figure 5-5 shows the information page that appears when the user opens one of the items in that sublist. (The information page in Figure 5-5 also appears when the user opens a stand-alone attribute created by dragging the item from the sublist and dropping it in a catalog or on the desktop.)

Figure 5-4 Information page with a sublist

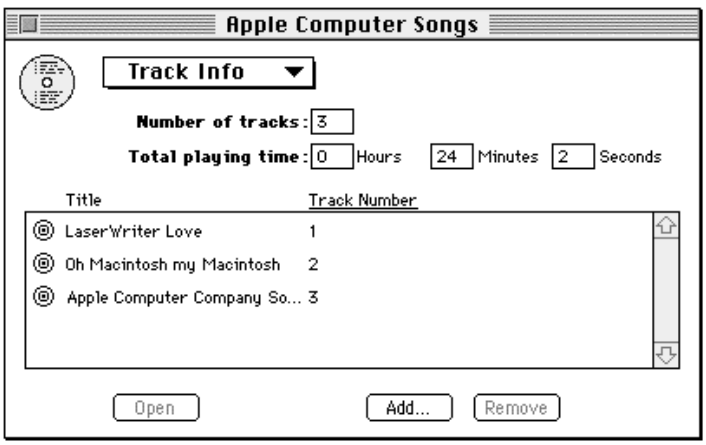


Figure 5-5 Information page for an item in a sublist

You can extend the ability of the AOCE Catalogs Extension to display the contents of records and attributes by providing AOCE templates.

Introduction to AOCE Templates

The AOCE Catalogs Extension to the Finder provided with the PowerTalk system software can display a certain number of record types and attribute types. If you want to provide users with the ability to work with other record types or attribute types, you can extend the AOCE Catalogs Extension (CE) by writing AOCE templates.

Templates allow developers to extend the browsing capabilities of the system in several ways: adding new types of visible records, defining the kind and category for a particular record type or attribute type shown in a list, and extending the available information pages for displaying the record or attribute contents to the user.

The data in a record is stored in data structures known as ***attributes***. Each attribute can contain any number of ***attribute values***. Each attribute has a *type*, and each attribute value has a *tag* that indicates its format. Attribute values can be up to 64 KB in size. The `attribute` structure is defined in the chapter "AOCE Utilities" in this book.

There is not necessarily a one-to-one correspondence between attribute values and data items of interest to a user. For example, the name and address of a person could be stored as a single attribute value, as two attribute values (one containing the name and one the address), or as several attribute values (one containing the first name, one the last name, one the house number and street, one the state, and so forth). There are no restrictions on the type of data that can be placed in attributes, and, except for a few standard attribute types, there is no way for the CE to determine how to display or interpret an attribute. For this reason, for each new record type or attribute type that you

add to a catalog, you must provide templates that tell the CE how to display the data contained in records or attributes of that type.

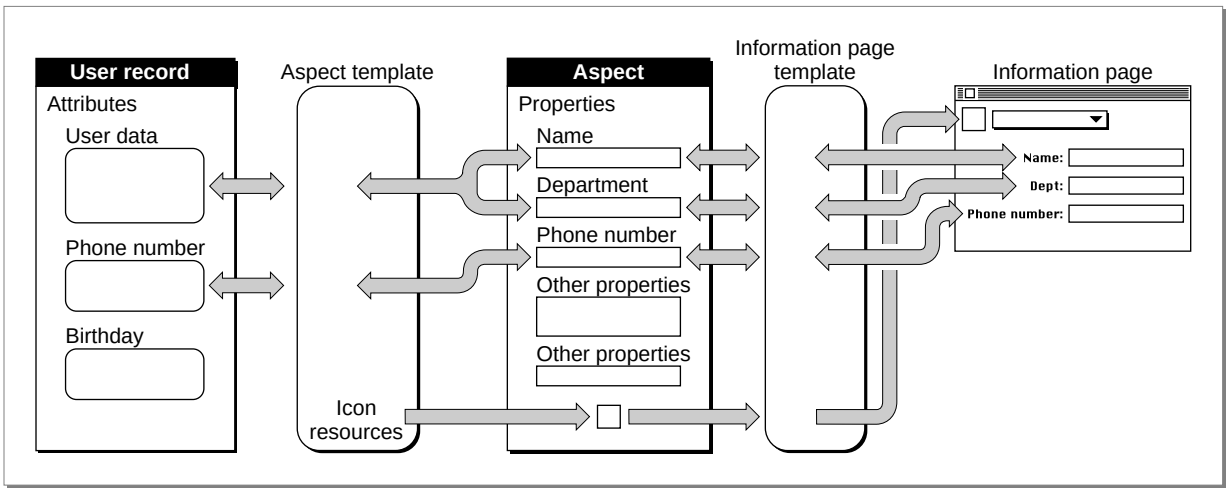
To display the data contained in records, AOCE templates must do two things: parse attribute values into the individual data items of interest to the user (referred to as ***properties***) and specify how the Finder should display each property. For example, an attribute value could contain three strings: a house number, a street name, and a city name. To display each string as a separate item on the screen, you would provide two AOCE templates: an aspect template and an information page template. The aspect template would specify that the house number, street name, and city name each constitutes a property, and the information page template would specify that each of these items is to be displayed in an editable text box and would specify the size and location of each text box in the information page.

Just as an information page template has an associated information page, an aspect template has an associated ***aspect***; a structure in memory that contains properties provided by the aspect template. Each information page displays properties taken from a single aspect. Note, however, that a given information page template need not use *all* of the properties in an aspect, and any number of information page templates can use properties from a single aspect.

Note
In the terminology of object-oriented programming, the information page is the view/controller and the attribute value is the persistent storage; the aspect template is the class for the aspect instance, and the information page template is the class for the information page instance. ◆

Figure 5-6 illustrates the relationships among records, aspect templates, aspects, information page templates, and information pages. As the figure shows, the aspect template processes the data in an attribute within a catalog record to create properties in an aspect. The aspect template itself might also provide data for properties, as is the case with the icon resources in the figure. The information page template specifies how the properties are displayed on an information page. The information page template can also provide such unchanging items as labels for text fields. Any editable item in the information page can also be processed in the other direction by the templates to change the data in the catalog record, as indicated by the bidirectional arrows in the figure.

Figure 5-6 From a record to an information page



There are five different types of AOCE templates:

- An ***aspect template*** specifies an aspect that provides information about a record or attribute of a particular type. Some of the information in an aspect is specified by the aspect template itself and therefore applies to all records or attributes of the same type. Examples of such information are the kind and category of an attribute or the icon of a record. Other information in the aspect is extracted by the template from the record, such as a string or number from the contents of an attribute value. Each aspect includes a collection of items of various types, which are known as properties. The aspect template includes instructions that tell the CE how to create properties from attributes and attributes from properties. Some aspect templates also specify how new records of a specific type are to be added to the containing dNode or how new attributes of a specific type are to be added to the containing record.

  Each aspect is independent of all other aspects, even aspects created from the same attribute or record. Therefore, two developers can provide separate aspect templates that act on the same attribute or record without causing any conflicts.

- An ***information page template*** specifies how the Finder should display record or attribute data. The information page template specifies which aspect to use to fill in the fields of the information page and the graphic layout of the information page.

- A ***forwarder template*** allows existing aspects and information pages to be used for new types of records. Using forwarders, a single information page template can be applied to several different record or attribute types.

- ***Killer templates*** allow developers to supersede existing templates. Killer templates identify one or more templates by name and cause the CE to ignore those templates. For example, if you want to override one of the built-in templates, you can provide a killer template that disables the existing template and a replacement template that the CE uses instead.
- In some applications of templates, all of the types of files that might contain templates cannot be known ahead of time. ***File type templates*** allow you to extend the list of file types that the CE searches for templates. All of the new files containing templates must, however, reside in the System Extensions folder.

The aspect and information page templates specify how to divide a record or attribute into properties and how to display and edit those properties through information pages. Forwarder, killer, and file type templates, in contrast, are concerned with which templates to use and where to find them.

Templates reside in the resource forks of files. The CE looks in several places for templates:

- template files (as indicated by a file type of `'detf'`) in the System Extensions folder
- MSAM and CSAM files in the System Extensions folder
- additional files of the types specified by any file type templates
- the PowerTalk Extension file, also in the System Extensions folder, which includes all the templates that are included as part of the PowerTalk software

Templates consist of a set of associated resources in the resource forks of these files. Each template has a ***signature resource*** that indicates the type of template. The ID of the signature resource is used to locate the other resources that make up the template. All additional resources are at fixed offsets from this base ID.

All templates, regardless of type, include a signature resource and a name resource. Each of the different types of templates also contain additional resources specific to that template's function. For example, an information page template includes the record and attribute types that it applies to, plus the layout of the information page to be displayed; a killer template includes a list of template names to be ignored.

Aspect Templates

An aspect template creates an aspect, which is a collection of information about a particular part of a record or an attribute. The information is divided into properties. Each property is identified by a unique number and can be any one of the following types:

Property type	Description
String	A string of text characters, stored internally as an `RString` structure
Number	A numerical value, stored internally as a long integer
Icon	An icon suite; always stored within the aspect template as a set of resources
Binary	An uninterpreted block of bytes, which can be used to store arbitrarily formatted data

Aspects serve two primary purposes:

- They provide unique identification for properties. Although within an aspect each property is identified only by a number and the same number is likely to be used for semantically different properties in different aspect templates, the combination of aspect template name and property number should be unique. To ensure that your aspect template has a unique name, you should start the template name with a 4-character application signature registered with Macintosh Developer Technical Support.
- They allow efficient access to subsets of the data in a record or attribute. For an item to appear in a sublist—such as a record in a dNode window or an attribute in a record information page—the Catalogs Extension must have icon, kind, and category information for that item. The CE takes this information from a single aspect. If you place other information about the item in other aspect templates, the CE does not create the other aspects for the item until they are needed, such as when an information page is opened for the item. Not creating aspects until they are needed saves time and memory. In addition, one aspect can often be used by more than one information page.

Property values in aspects are derived from three sources:

- the aspect template itself
- the record or attribute value
- the CE, which fetches related information not directly a part of an attribute (such as the access mask)

Some properties provide unchanging information, such as a record's kind or default values for changeable information. The CE takes these properties from resources in the aspect template itself by using the property number as an offset from the base ID of the template. For example, a resource of type `'rstr'` (`RString`) with an ID of 1013 in a template with a base ID of 1000 corresponds to a string property with property number 13.

For properties that must be taken from a record or attribute, the aspect template includes directions for dividing up attribute values to extract the properties. The aspect template

can include a lookup table with instructions for parsing attribute values into properties (“The Lookup-Table Resource” beginning on page 5-105), a code resource (“Code Resources Reference” beginning on page 5-142), or both. The CE uses the same process in reverse to revise or create attribute values in the record when the user edits a field in an information page. The lookup-table mechanism for parsing attribute values should allow you to create most of your aspect templates without writing any code. Aspect template lookup tables (also referred to as *patterns*) provide a wide range of different types of data structures that can be combined to handle almost any attribute format.

An aspect template can convert data from one type to another as it divides an attribute value into properties and also when it takes the value of a property from a field on an information page. For instance, an aspect template might convert a number in an attribute to a string property to allow the user to edit it. The property-type system is extensible, allowing the aspect template code resource to supply additional types and perform the appropriate type conversions.

Each aspect template includes a specification of the types of records and attributes to which the template applies. An aspect template can apply to a particular type of record or to a particular type of attribute found either in any record type or only in specific record types.

Information Page Templates

There is an information page template for each information page displayed to users. The template specifies the physical layout of the information page and indicates what properties are used to fill in each field (or ***view***) in the information page.

For a list of possible view types, see “View Lists” on page 5-123.

Like aspect templates, information page templates apply only to a specified record or attribute type.

Forwarder Templates

Forwarder templates allow a new record or attribute type to use existing aspect and information page templates. A forwarder template includes a specification of the record type and attribute type to which it applies, just as is the case with aspect and information page templates. In addition, the forwarder template contains a list of aspect and information page template names to be used with the specified record or attribute type.

Killer Templates

Killer templates disable existing templates. A killer template consists of a list of names of the templates to be disabled.

Killer templates do not change the affected template. They just render it inactive at the time it would have been used.

You can use killer templates to disable any type of template except another killer template.

File Type Templates

The CE looks for templates during system initialization and the first time the CE needs a template after someone has called the `kDETcmdUnloadTemplates` callback routine (page 5-208). The CE always looks for templates in files of type 'detf', `'dsam'`, `'msam'`, and `'csam'`; and in files of type `'fext'` that have creator type `'adbk'`. (See "File and Resource Types Used by the Catalogs Extension" on page 5-73 for more information about these file types.) File type templates specify additional file types in which the CE looks for templates. The new files can also include file type templates. All of the new files containing templates must reside in the System Extensions folder.

How Aspect and Information Page Templates Work

The most important function of AOCE templates is to provide users with new information pages through which they can view and modify information contained within the AOCE catalog system. To accomplish this, the Catalogs Extension starts with a record or attribute and, following the instructions in aspect templates, creates aspects of the record or attribute. Each aspect can contain information derived from a single attribute, or—in the case of aspects derived from records—from several attributes. Because each aspect is independent from other aspects of the same record or attribute, you can create them without concern for what other developers might do. An aspect contains one or more values, each of a specific type—a text string or a number, for example. These values are referred to as *properties*. The CE then follows the instructions in an information page template to use the properties to fill in one or more fields (referred to as *views*) in the information page that is displayed to the user. Each information page is associated with only one aspect, from whose properties values for all of its views are taken.

When the user changes a value in an information page and then closes the page, the CE uses the same process in reverse to revise the attribute value in the record. If the record does not already contain this attribute value, the CE creates a new one from the values in the information page, following the instructions in the aspect template.

Figure 5-7 shows the process of translating between a record and an aspect. The aspect template uses lookup tables and code resources to parse the attribute data into properties. The aspect template can also contain property resources that it uses to create uneditable properties; in Figure 5-7 the icon property is created in this way.

Figure 5-7 Creating an aspect from a record

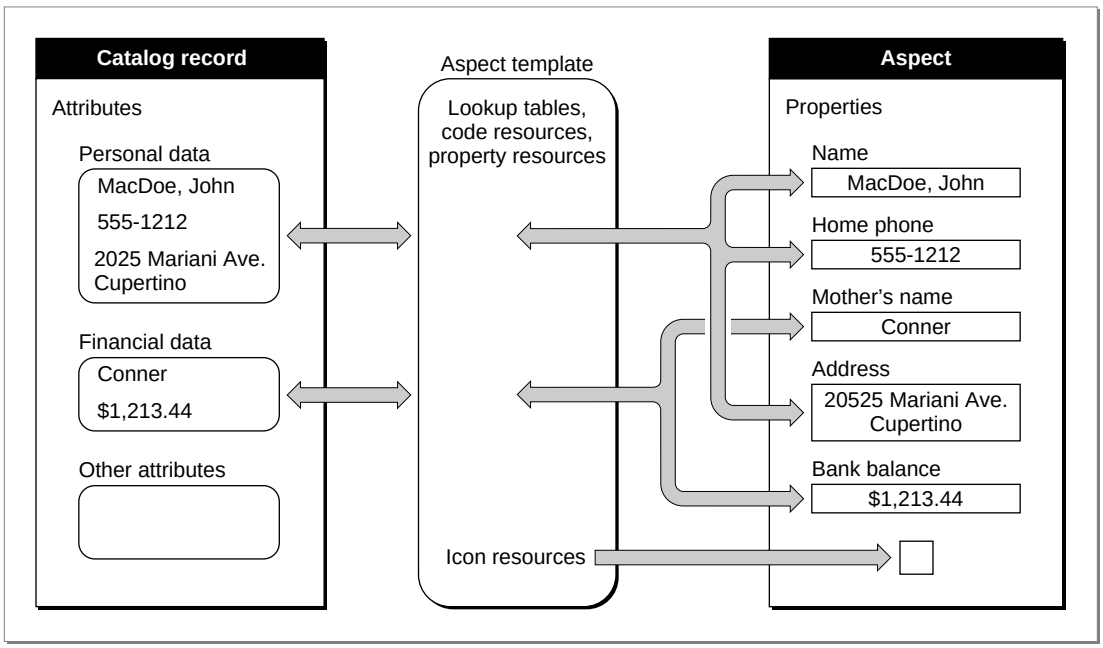
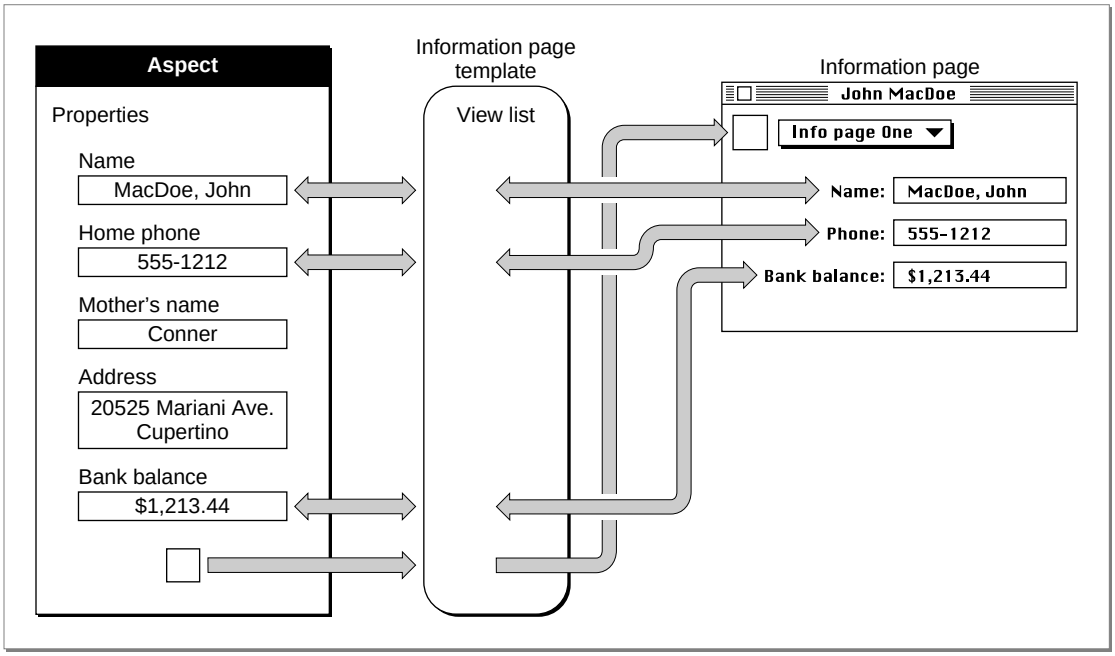


Figure 5-8 shows how the information page template translates properties in an aspect into views (fields) in an information page. The information page template includes one or more view lists that describe the layout and type of each field in the information page and that assign a property to each view. Note that not all the properties in the aspect must be represented in a single information page.

Figure 5-8 Creating an information page from an aspect

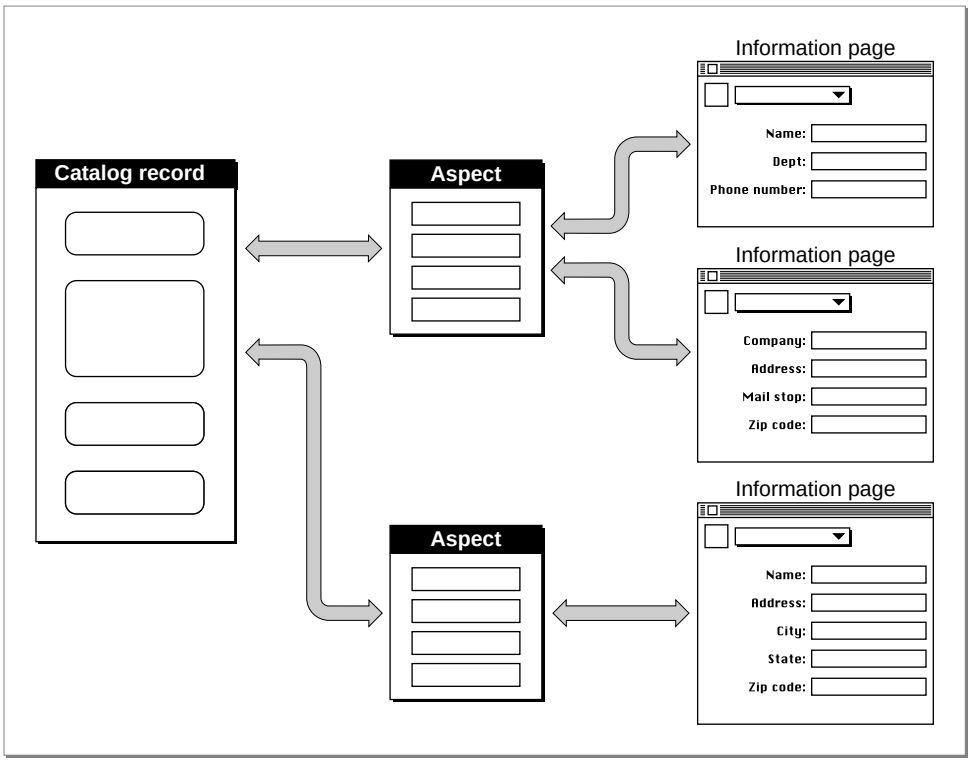


Any number of aspect templates can extract data from the same record or attribute, and the aspects they create can exist simultaneously without conflict. Because the properties in a given aspect are identified only by number, the CE uses the combination of the aspect's name and the property number to identify a property uniquely. Therefore, it is important that the name of an aspect be unique. When you create an aspect template, you should give it a name that includes a description of its purpose plus your application signature or other unique identifying string. The CE uses aspect names solely for internal identification; the user never sees them.

An information page takes the properties for all of its views from a single aspect. Each view specification in the information page template tells which property number is to be used. The information page template itself includes the name of the aspect within which these properties are found.

Figure 5-9 shows a record with multiple aspects and information pages. Note that one aspect can be used by more than one information page.

Figure 5-9 Multiple aspects and information pages



When a record or attribute appears in a sublist—in a dNode window in the case of records, or in an information page in the case of attributes—the CE takes the properties needed to fill in the data for an item in the list from a special aspect known as the ***main aspect***. Whereas any aspect can contain information about the *contents* of a record or attribute, only a main aspect contains information about the record or attribute itself: its name, kind, category, and icon. In the case of records, this information usually consists of unchangeable properties stored permanently in the main aspect template. In the case of attributes, however, the main aspect template often retrieves the information from the attribute. Thus, the information changes when someone edits the attribute value.

There is a separate main aspect for each item in a sublist (note, however, that more than one of these main aspects may be derived from the same aspect template). Figure 5-10 shows how main aspects for records are used to fill in the contents of a sublist in a dNode window.

Figure 5-10 Main aspects for records

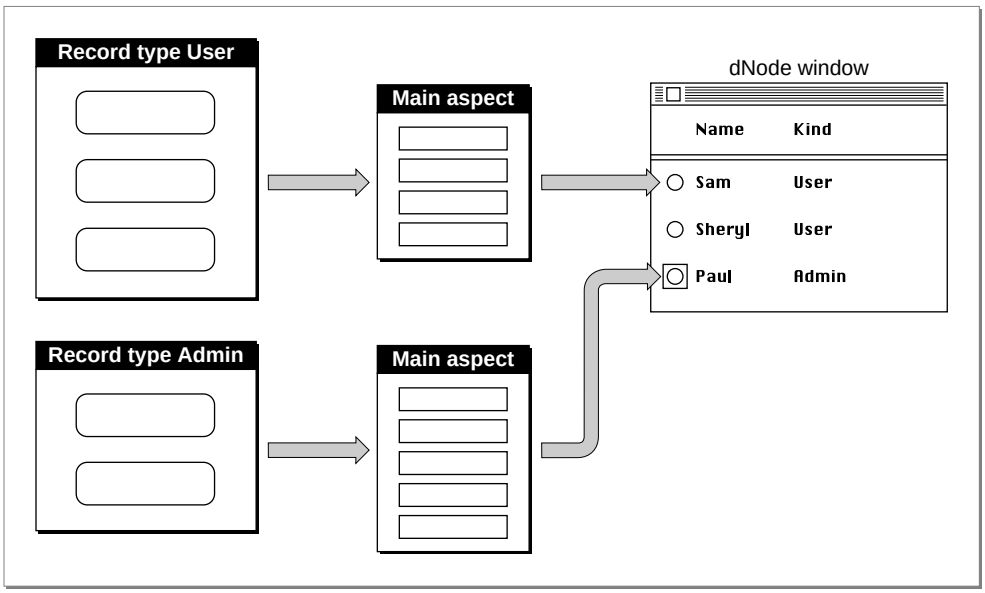
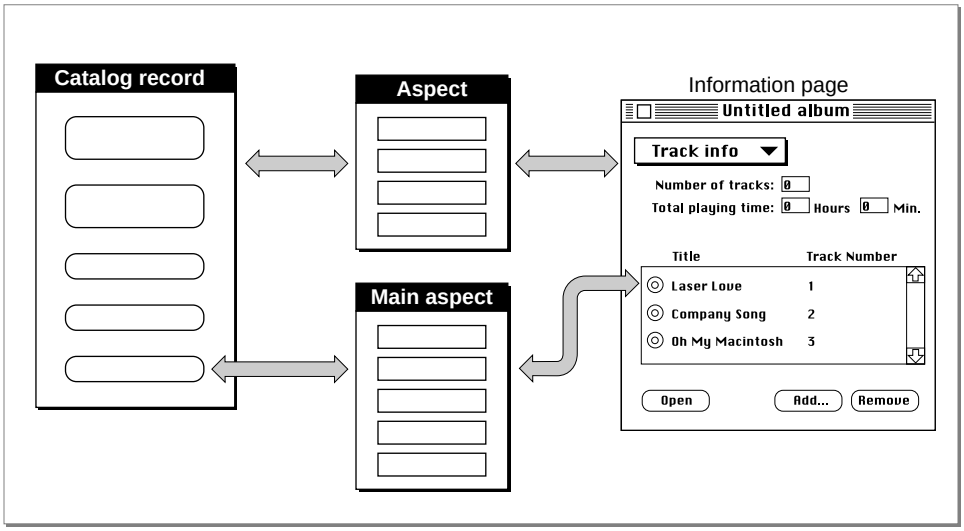


Figure 5-11 shows how aspects and main aspects are used to fill in the contents of an information page that contains a sublist. All of the properties for the views in the main part of the information page come from a single aspect, the ***main view aspect***. This aspect also specifies whether there is a sublist in the information page and which attribute types are to be included in the sublist. Each attribute in the sublist has its own main aspect, which provides the information shown in the sublist for that attribute.

Figure 5-11 shows two of the aspects used to fill in an information page: the main view aspect and a main aspect for an item in the sublist. There is one aspect template for each attribute type in the sublist, and a separate aspect template for the main view aspect. Note that, whereas the properties in the aspect for the main part of the information page can come from any number of attributes in the record, a main aspect (which describes a single line in a sublist) derives its properties from a single attribute value.

Figure 5-11 Main aspects for attributes



There must be a separate main aspect template for each type of record displayed in a dNode window and for each type of attribute displayed in a sublist. The CE can use a main aspect template to create main aspects for any number of items of the same type. Figure 5-12 shows three records, two of type User and one of type Admin. The two records of type User are processed by a single main aspect template to create a main aspect for each record; the record of type Admin is processed by a separate main aspect template to create its main aspect.

Figure 5-12 Main aspect templates for records

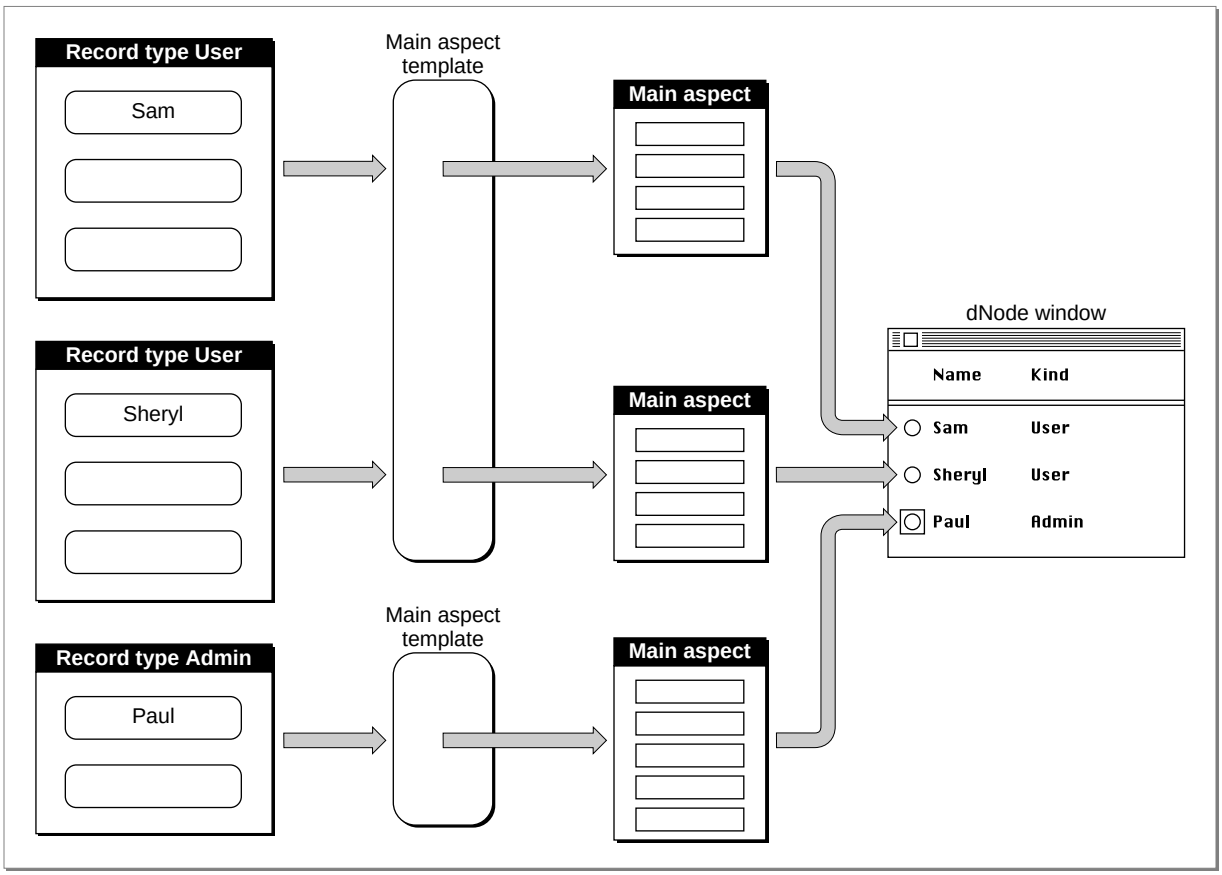
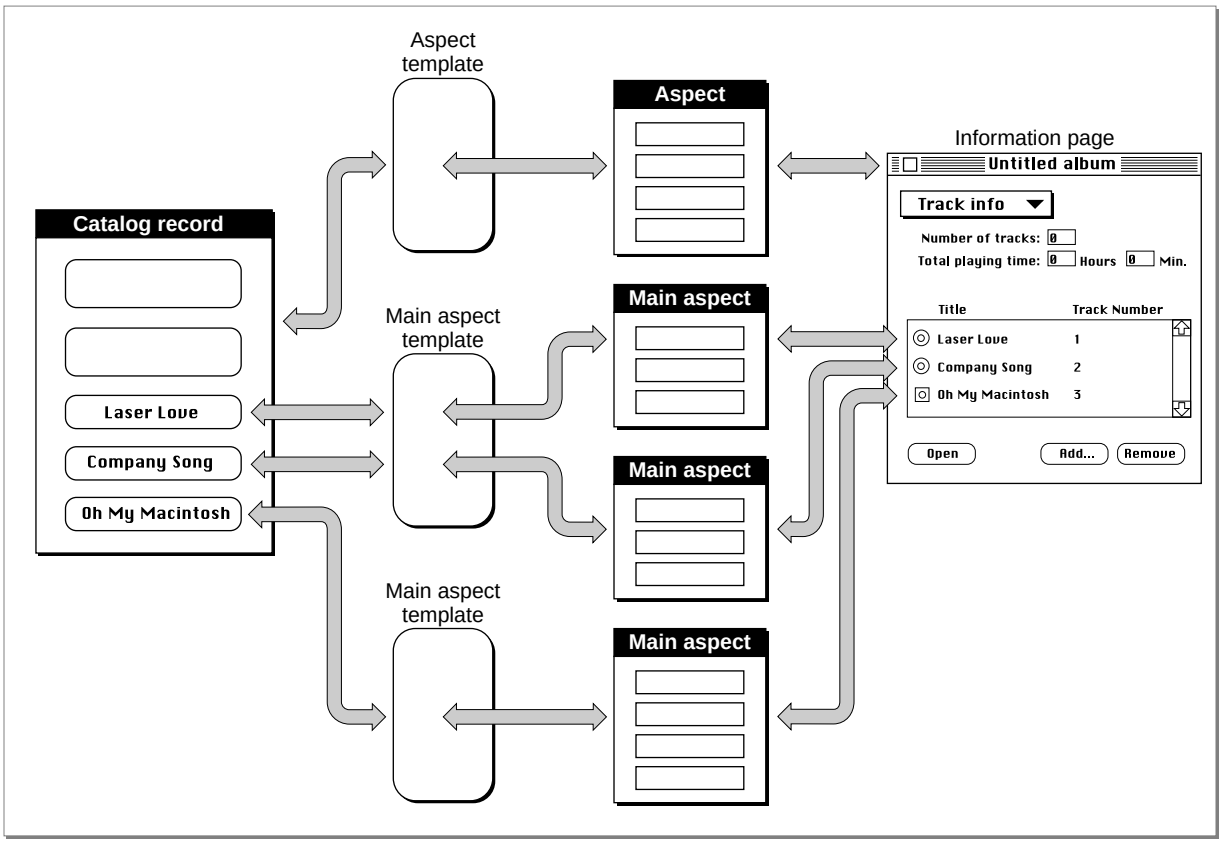


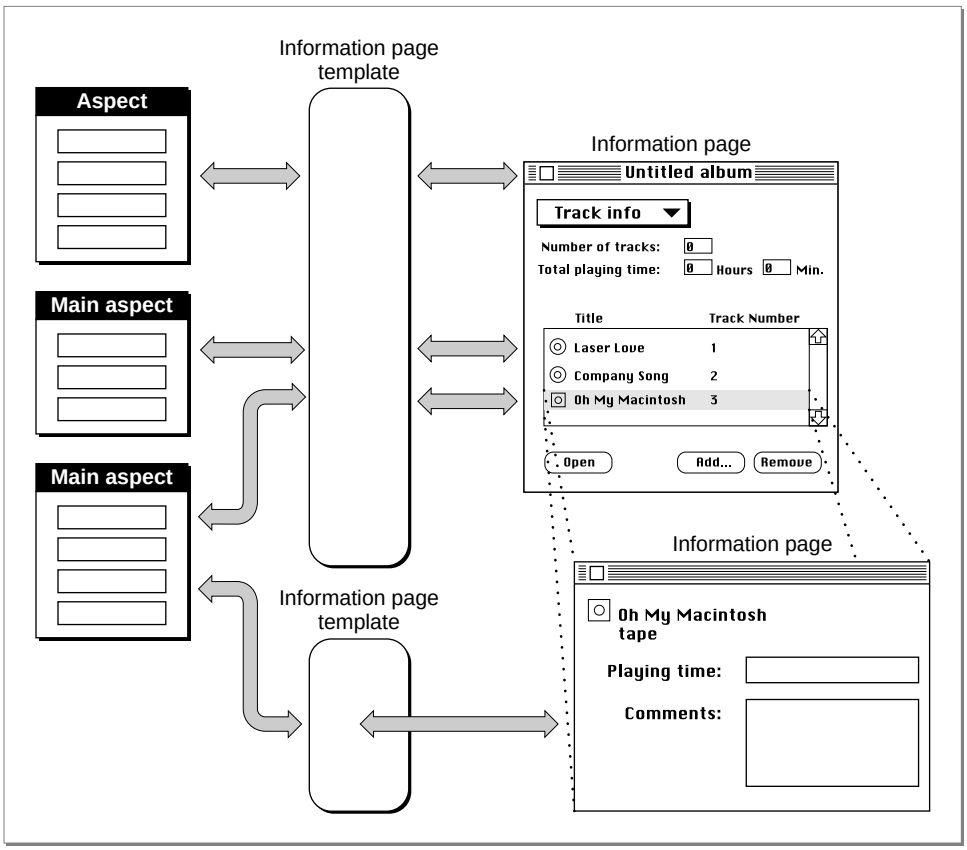
Figure 5-13 shows how an information page with a sublist is created from one aspect template and one or more main aspect templates. The aspect template creates an aspect for the main part of the information page. Each attribute type has a separate main aspect template; several attributes of the same type might be processed by the same main aspect template. Each attribute in the sublist has its own main aspect.

Figure 5-13 Main aspect templates for attributes



A main aspect can contain properties used by other information pages as well as the information needed for a sublist. As shown in Figure 5-14, a typical use for this feature is for a main aspect to contain all the properties for the information page that appears when the user double-clicks an attribute in a sublist. Note that all of the views for a single information page are described in a single information page template. Even the position of the sublist and the layout of each line in the sublist are described in this information page template. As shown in the figure, the information page that appears when the user opens an attribute in the sublist requires its own information page template.

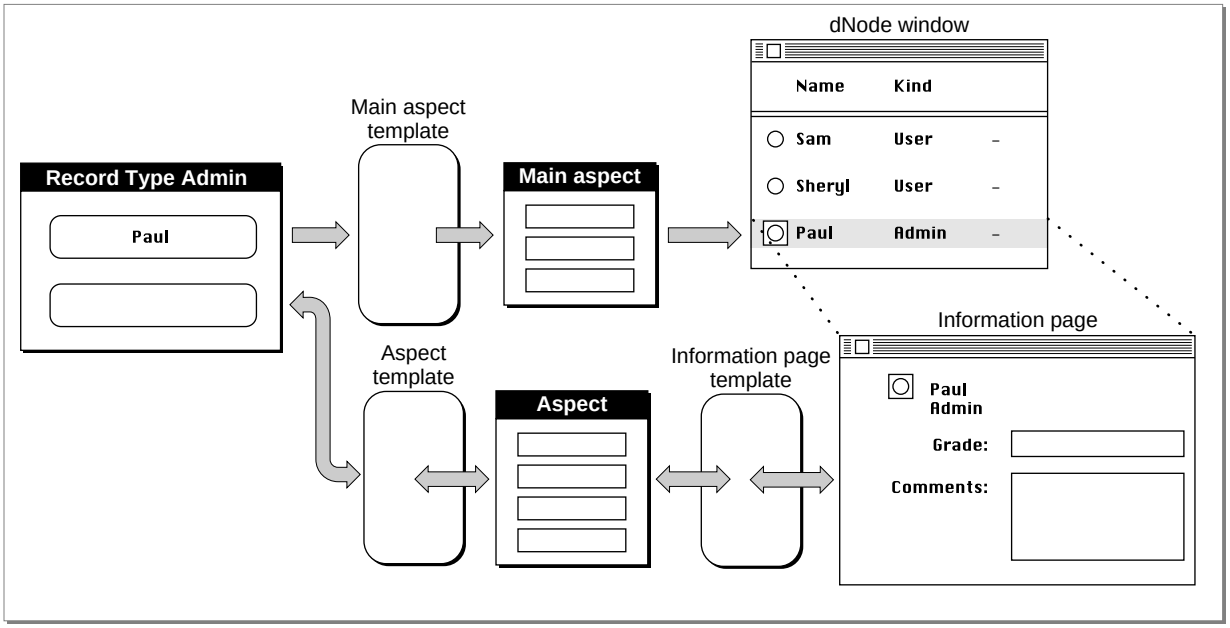
Figure 5-14 Providing an information page for an attribute in a sublist



In contrast to the situation shown in Figure 5-13 and Figure 5-14, a record shown in a dNode window list typically has at least two aspect templates associated with it: a main aspect template used to display information about the record in the dNode window plus one or more aspect templates used to provide properties for the information pages that are displayed when the user opens that record. Figure 5-15 illustrates this situation. Note that you must provide both an aspect template and an information page template to display the contents of the record, but you do not provide an information page template for the dNode window.

Keeping the main aspect template and other aspect templates for a record separate allows the Catalogs Extension to load into memory only the aspects that are needed at a given time and makes it easier for developers and users to create new information pages for an existing record type.

Figure 5-15 Providing an information page for a record in a dNode window list



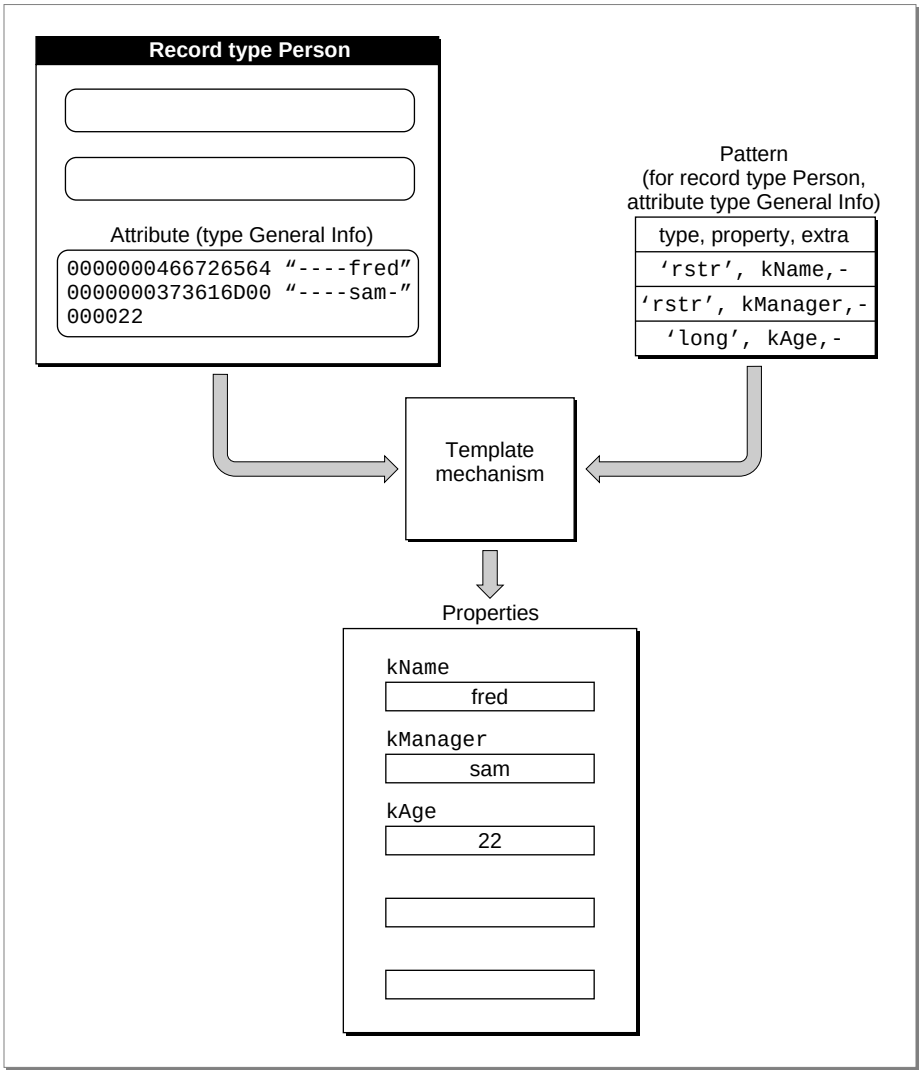
A main aspect template for a record type specifies how new records of that type are to be added to the containing dNode. Similarly, a main aspect template for an attribute type specifies how new attributes of that type are to be added to the containing record.

Main aspect templates are described in "Aspect Template Signature Resource," which starts on page 5-88.

The process of filling in views in information pages from properties in aspects is fairly straightforward. The information page template includes a property number for each view that requires data from the aspect. The information page template includes instructions for how to interpret the contents of the property—as a number, a text string, and so forth—how to display it, and whether to allow the user to edit it. Information page templates may display the same type of property in different ways depending on the circumstances. For example, a number property is used in both checkboxes and pop-up menus. In checkboxes, this property indicates whether the checkbox is selected or not. In pop-up menus, it indicates which of the entries in the menu is currently selected.

The process of filling in properties from records and records from properties is more complex. The aspect template provides two mechanisms: lookup tables and code resources. Lookup tables can translate a large variety of data structures without requiring you to write any code. Code resources cover all data formats, including those not handled by lookup tables. In addition, code resources can perform actions based on the new data being written to the catalog system. For example, adding a new user record might trigger an update of a personal gateway's internal user list.

Figure 5-16 Pattern-based attribute parsing



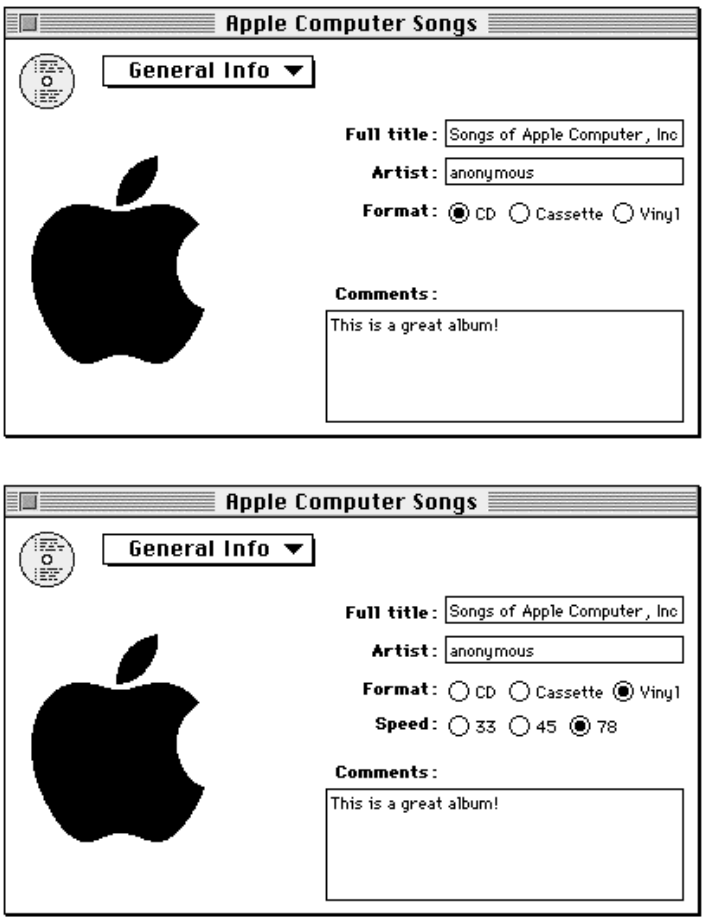
Lookup Tables

A ***lookup table*** contains a pattern that describes the contents of an attribute and specifies into which property to store each part of the attribute value. In many cases, the pattern is extremely simple. For instance, an attribute value might consist only of a single string of type `RString` or only of a single binary number. Other attribute value formats may be more complex, combining multiple items in a single attribute value, or requiring conditional evaluation of the contents. Figure 5-16 illustrates the basic process of creating properties from attribute values.

A lookup table also works in reverse, revising the contents of attribute values when the user enters new data in views in the information page. If an attribute value does not

already exist, the lookup table creates a new attribute value from the data and puts it into the record.

Figure 5-17 Conditional view



Conditional Views

The information page template includes one or more view lists; each view list describes one or more views that can be displayed on the information page. Each view list is associated with two property values. The views described by that view list are displayed only if those property values are equal (or if either property equals `kDETNoProperty`). Therefore, you can control whether a particular view is displayed on the information page by changing the values of properties in the aspect associated with the information page. Views that are made to appear and disappear in this fashion are called ***conditional views***. Figure 5-17 illustrates the use of a conditional view. In this case, the radio buttons that specify the playing speed of the album (33, 45, or 78 RPM) appear only when the

user selects Vinyl as the format of the album. For more information on conditional views, see Listing 5-14 on page 5-122 and "Implementing Conditional Views" beginning on page 5-131.

Code Resources

Aspect templates can include code resources that allow developers to extend the capabilities of the templates.

The Catalogs Extension calls your code resource when certain events occur that affect the aspect with which the code resource is associated. Such events include user actions, such as the user clicking a button in an information page or dropping a file on a catalog object, and administrative events, such as initialization or a query as to whether a control should be drawn as enabled. The code resource may call the CE to perform a variety of services, such as returning information, converting one data type to another, or updating an information page.

The routine selectors and parameters that the CE passes to your code resource are described in "Functions You Can Provide as Part of Your Code Resource" beginning on page 5-148. The CE-provided routines that your code resource can call are described in "CE-Provided Functions That Your Code Resource Can Call" beginning on page 5-196.

How the Catalogs Extension Saves New Values

When the user closes an information page or makes another information page the active one, the Catalogs Extension checks all of the visible views in the information page the user just closed or left. If the user has changed any of the properties associated with those views, the Catalogs Extension saves the new values.

For each property, the CE first calls the code resource for the aspect from which the property came with the `kDETcmdValidateSave` routine selector (page 5-168). If your code resource does not return an error for any of the changed properties, then for each of these properties, the CE finds all of the lookup-table patterns that include that property. The CE processes those lookup-table patterns to write attribute values. Any of the lookup-table patterns can contain custom elements that you define; in that case, the CE calls your code resource to process those elements.

For a property to be saved, it must both be in a visible view and be marked as changed, or it must be in a lookup-table pattern with another property for which those two conditions are met.

If the user makes a change to a sublist value, the CE saves the change as soon as the user leaves the item and clicks somewhere else on the screen. The CE uses the lookup-table pattern in the main aspect for items of the type changed to process the change. The CE does not call your `kDETcmdValidateSave` routine for changes in sublists.

Property Value Synchronization

The Catalogs Extension checks a catalog system flag periodically to see if the data in the catalog system has changed. If it has, the Catalogs Extension processes the lookup tables of all the aspects for open information pages, recalculating all the properties derived from the catalog system. The CE then updates the aspects and open information pages accordingly. At the time the CE checks for changes, it calls the aspects' code resources with the `kDETcmdShouldSync` routine selector (page 5-185). If you have derived any properties from data outside the catalog system or from records or attributes other than the one to which your aspect applies and you have reason to believe their values have changed, your code resource should tell the CE to update all the properties, which it will then do whether data in the catalog system has changed or not.

When the CE updates all the property values in an aspect—either because data in the catalog system has changed or because your code resource told it to—the CE calls your code resource with the `kDETcmdDoSync` routine selector (page 5-186). If your code resource has supplied any of the property values, you should update your sublist items and your other properties.

When the CE synchronizes a sublist, it first marks every item in the list as "unseen." The CE then reads in all the attribute values mentioned in the lookup tables and calls the code resource's `kDETcmdDoSync` routine. The code resource should update any sublist items that it supplied. For each attribute the CE processes that the lookup table lists as for use in the sublist, the CE checks the type and creation ID of the item to see if it is already in the sublist. If the item is in the sublist, the CE updates it and marks it as "seen." If it's not there, the CE creates a new item, adds it to the sublist, and marks it as "seen." After processing all such items, The CE removes from the sublist any items that are still marked "unseen."

Drags and Drops

The user can drag HFS and catalog objects—such as files, information cards, records, and attributes—and drop them on records, attributes, or sublists. In each case, the Catalogs Extension determines the most appropriate action based on the type of object dragged, the type of object on which the item was dropped, and instructions in the aspect templates of the dragged and destination objects (see note at end of this section).

For example, if the user drags an information card and drops it on a record in a catalog, the CE checks every aspect template available that applies to that record for resources that provide drop instructions. The CE then determines what to do (perhaps to add an alias to the information card to the sublist of the record) and calls the code resource (if any) in each aspect for the record. The code resource can take some other action, carry out the action recommended by the CE, or take no action and return control to the CE. See the descriptions of the `kDETcmdDropQuery` (page 5-172) and `kDETcmdDropMeQuery` (page 5-170) routines for more information on how code resources handle drags and drops.

If the user drags more than one item onto a catalog object, the CE collects all of the operations and executes them in batches—for instance, the CE might copy half the items and use the other half to invoke custom operations in a destination code resource.

In addition, there may be more than one aspect in the destination that can accept a drop. For example, a Group record includes an aspect that can add a user to the group and another aspect that can mail an information card to a group. Each aspect includes a string that the CE can present to the user to confirm the action.

The aspect template signature resource includes a drop-check Boolean value and a drop-operation order number. If there is only one aspect that can handle the drop and you specify `dropCheckAlways` as the Boolean value, the CE displays a dialog box to let the user confirm the action. You must provide the prompt string for the dialog box in an aspect template resource. If you specify `dropCheckConflicts` as the Boolean value, the CE handles the drop without checking with the user. If there is more than one aspect that can handle the drop, the CE displays a confirmation dialog box for the option offered by the aspect that has the lowest drop-operation order number.

The resources that you must provide in an aspect template to support drags and drops are described in "Supporting Drags and Drops" beginning on page 5-98.

How the Catalogs Extension decides which drop operation to perform

The process the Catalogs Extension goes through to decide which drop operation is appropriate is fairly complex. First, the CE finds every aspect that might accept the drop (that is, every aspect of the destination object that has drag-in resources or a code resource). For each one, the CE figures out what operation the aspect wants to perform by looking at where the aspect is located (for example, whether to move an object or copy it depends on whether the destination is on the same volume as the original location), the access masks (can the CE delete the original, for example?), the drag-in and drag-out resources in the aspects of the source and destination containers, and the code resource (if any).

The CE calls the code resource in the aspect of the object being dropped with the `kDETcmdDropMeQuery` routine selector and then calls the code resource of the destination aspect with the `kDETcmdDropQuery` routine selector. These routines can specify that a different action be performed in response to the drop. In both bases, if the code resource does not handle the request, the CE calls the code resource of the object's container (if the object is an attribute, its container is a record).

At this point, the CE has a list of possible operations—one for each possible destination. If the user has dragged several objects, the CE repeats this process until it has such a list for each item being dropped. Then the CE groups together all the items that share the same set of possible operations. For each group for which there's a choice of possible operations, the CE selects the operation with the lowest drop-operation order number and displays a dialog box asking the user whether to perform the operation.

The operation can be a move, a copy (also referred to as a drag), the creation of an alias, or the sending of a property command to a code resource. If the operation is a property command specified by the destination's code resource (in response to the `kDETcmdDropQuery` request), then the CE sends the property command to the destination's code resource. If the operation is a property command specified by the dragged object's code resource (in response to the `kDETcmdDropMeQuery` request), then the CE sends a property command to that code resource. If the operation is a move, copy, or creation of an alias, then the CE carries out the operation itself, displaying status windows as appropriate. ◆

Writing AOCE Templates

This section provides some simple examples of source code for AOCE templates. The templates shown here create a new record type that stores information about a user's collection of recording albums. A user can place these templates in the System Folder to add a new record type and information pages to his or her personal catalog.

A set of AOCE templates includes a large number of resources of several different types. To understand this section you must be familiar with the definitions and concepts provided in the preceding sections of this chapter. In addition, the resource types used in this section are all described fully in "AOCE Templates Reference" beginning on page 5-73 and "Code Resources Reference" beginning on page 5-142; cross references to the reference material are provided wherever practical. You will probably have to refer to the reference material frequently while reading this section. Additionally, the AOCE templates provide many features not illustrated by these examples; to learn about all these features you will have to read the reference sections in detail.

Note
All of the resource examples in this chapter are written in the syntax of the Rez resource compiler. All other code is written for the MPW C compiler. ◆

Defining a New Record Type or Attribute Type

When you define a new record type or attribute type, you must provide a main aspect for that record or attribute type. The main aspect includes a signature resource, a resource that specifies the record or attribute type to which the main aspect applies, and several other resources (for instance, icon resources, the text of help balloons, the text of the New item in the Catalogs menu for records or the text of the Add item in the new-attribute-item dialog box, and the name given to newly created records or the initial value for new attributes). For a full list of the required and optional resources used only by main aspect templates, see Table 5-4 on page 5-89. Additionally, main aspect templates can contain any of the resources found in other aspect templates. For a full list of resources that can be used by aspect templates, see Table 5-1 on page 5-78.

Listing 5-1 shows a main aspect template for a new record type. Because Listing 5-1 is for a main aspect template for a record, it includes only resources that are specific to the main aspect. Separating the main aspect template from other aspect templates for a record has certain advantages. This segregation of resources into main aspect templates and other aspect templates allows the Catalogs Extension to load into memory only the aspects that are needed at a given time and makes it easier for developers and users to create new information pages for an existing record type.

Note
In order to ensure uniqueness of attribute and record types, this and other code listings in this chapter use WAVE, the application signature of the fictitious application SurfWriter, as the first part of all attribute and record type names. ◆

Listing 5-1 Main aspect template

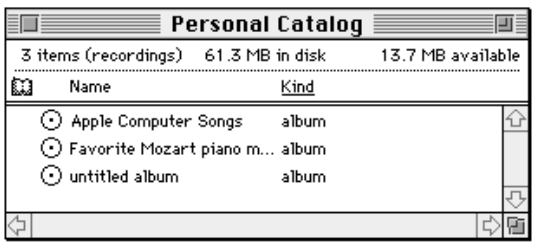
```
// File: AlbumMainAspect.r

#include "Types.r"
#include "OCETemplates.h"
#include "OCE.r"

#define kAlbumMainAspect   kDETFirstID

// Aspect template signature resource

resource 'deta' (kAlbumMainAspect, purgeable) {
   0,                   // drop-operation order (not used in this aspect)
   dropCheckAlways,     // drop-check flag (not used in this aspect)
   isMainAspect         // is the main aspect
};

// Template name

resource 'rstr' (kAlbumMainAspect + kDETTemplateName, purgeable) {
   "WAVE Album Main Aspect"   // start name with application signature
};

// Record type to which this template applies

resource 'rstr' (kAlbumMainAspect + kDETRecordType, purgeable) {
   "WAVE Album"   // start with application signature
};
```

```
// Categories to which this record type belongs

resource 'rst#' (kAlbumMainAspect + kDETAspectCategory,purgeable)
   {{
   "Recordings"
   }};

// String to be displayed in the Catalogs menu

resource 'rstr' (kAlbumMainAspect + kDETAspectNewMenuName, purgeable) {
   "New Album"
};

// Name given to new record of this type

resource 'rstr' (kAlbumMainAspect + kDETAspectNewEntryName, purgeable) {
   "untitled album"
};

// Record kind as shown in a sublist

resource 'rstr' (kAlbumMainAspect + kDETAspectKind, purgeable) {
   "album"
};

// Text for help balloons

resource 'rstr' (kAlbumMainAspect + kDETAspectWhatIs, purgeable) {
   "Album\n\nA description of an album. Open this icon to display information
    about the album."
};

resource 'rstr' (kAlbumMainAspect + kDETAspectAliasWhatIs, purgeable) {
   "Album alias\n\nAn alias to a description of an album. Open this alias to
    display information about the album."
};

// Icons

include "AlbumIcons" 'ICN#'(0) as
      'ICN#'(kAlbumMainAspect + kDETAspectMainBitmap, purgeable);
include "AlbumIcons" 'icl4'(0) as
      'icl4'(kAlbumMainAspect + kDETAspectMainBitmap, purgeable);
```

```
include "AlbumIcons" 'icl8'(0) as
      'icl8'(kAlbumMainAspect + kDETAspectMainBitmap, purgeable);
include "AlbumIcons" 'ics#'(0) as
      'ics#'(kAlbumMainAspect + kDETAspectMainBitmap, purgeable);
include "AlbumIcons" 'ics4'(0) as
      'ics4'(kAlbumMainAspect + kDETAspectMainBitmap, purgeable);
include "AlbumIcons" 'ics8'(0) as
      'ics8'(kAlbumMainAspect + kDETAspectMainBitmap, purgeable);
include "AlbumIcons" 'SICN'(0) as
      'SICN'(kAlbumMainAspect + kDETAspectMainBitmap, purgeable);
```

The main aspect in Listing 5-1 makes it possible for the user to create a new record of type Album by choosing New Album from the Catalogs menu. A new record of this type has the name "untitled album" until the user renames it. Records of this type are displayed in the catalog window when the user chooses Recordings from the View menu. Figure 5-18 shows a personal catalog window displaying records of type Album. The icons displayed in this window are provided by the icon resources in the main aspect.

Figure 5-18 Catalog window displaying the record type defined by Listing 5-1



Defining the Contents of the New Record Type or Attribute Type

When you define a new record type or attribute type, you must define its contents and provide a mapping between attributes and properties. In the case of a new attribute, you would normally include this information in the main aspect template. In the case of a record, however, you usually provide a separate aspect template to support each information page.

Listing 5-2 shows an aspect template for the Album record type defined in Listing 5-1. This aspect template defines several properties, provides a lookup table mapping attributes to properties, and provides default values and help-balloon strings for each property type. The lookup table maps a single attribute (`"WAVE Album General Info"`) into four properties (`prArtist`, `prTitle`, `prComments`, and `prFormat`) and a second attribute ("WAVE Album Cover") into another property (`prCover`). Note that this mapping also works in reverse: The first time the user provides new values for the properties and closes the information page, the Catalogs Extension creates the attributes

and places them in the record. Lookup tables are described in "The Lookup-Table Resource" beginning on page 5-105.

Listing 5-2 Defining properties for a record

```
/*
   File:    AlbumMainAspect.r
*/

#include "Types.r"
#include "OCETemplates.h"
#include "OCE.r"

#define kAlbum1stInfoPageAspect  kDETSecondID

// Aspect template signature resource

resource 'deta' (kAlbum1stInfoPageAspect, purgeable) {
   0,                   // drop-operation order (not used in this aspect)
   dropCheckAlways,     // drop-check flag (not used in this aspect)
   notMainAspect        // not the main aspect
};

// Template name

resource 'rstr' (kAlbum1stInfoPageAspect + kDETTemplateName, purgeable) {
   "WAVE Album First Info Page Aspect" //start with application signature
};

// Record type to which this template applies

resource 'rstr' (kAlbum1stInfoPageAspect + kDETRecordType, purgeable) {
   "WAVE Album"                        //start with application signature
};

// Properties

#define prTitle      kDETFirstDevProperty
#define prArtist     kDETFirstDevProperty + 1
#define prComments   kDETFirstDevProperty + 2
#define prFormat     kDETFirstDevProperty + 3
#define prCover      kDETFirstDevProperty + 4
```

```
// Lookup table - maps attributes to properties

resource 'dett' (kAlbum1stInfoPageAspect + kDETAspectLookup, purgeable) {
   {
   {"WAVE Album General Info"}, typeBinary,
      useForInput, useForOutput, notInSublist, isNotAlias, isNotRecordRef,
      {
         'rstr', prTitle, 0;
         'rstr', prArtist, 0;
         'rstr', prComments, 0;
         'word', prFormat, 0;
      };
   {"WAVE Album Cover"}, typeBinary,
      useForInput, useForOutput, notInSublist, isNotAlias, isNotRecordRef,
      { 'rest', prCover , 0 };
   }
};

// Default property values

resource 'rstr' (kAlbum1stInfoPageAspect + prTitle) {
   "<Put the album’s full title here.>"
};

resource 'rstr' (kAlbum1stInfoPageAspect + prArtist) {
   "<Put the album’s recording artist or group here.>"
};

resource 'rstr' (kAlbum1stInfoPageAspect + prComments) {
   "<Put comments here. Did you like it? What's the best track?>"
};

resource 'detn' (kAlbum1stInfoPageAspect + prFormat) {
   1
};

include "AlbumIcons" 'detb'(0) as
   'detb'(kAlbum1stInfoPageAspect + prCover, purgeable);


// Text for help balloons for the properties

resource 'rst#' (kAlbum1stInfoPageAspect + kDETAspectBalloons,purgeable) {
```

```
   {
   "The full title.", "The full title. Uneditable because the record is
    locked or access is restricted.",
   "The artist or group.", "The artist or group. Uneditable because the
    record is locked or access is restricted.",
   "Comments.", Comments. Uneditable because the record is locked or
    access is restricted.",
   "Format.", "Format. Uneditable because the record is locked or
    or access is restricted."
   "Album's cover.", Album's cover. Uneditable because the record is locked
    or access is restricted."
   }
};
```

To display the properties defined in Listing 5-2, you must provide an information page.

Laying Out an Information Page

Once you have defined a new record type or attribute type, or even if you just want to display the contents of an existing record type or attribute type in a new way, you have to provide one or more information page templates that tell the Catalogs Extension how to display the information in the record or attribute.

Listing 5-3 provides an icon and title for the information page and lays out the way in which the properties are displayed. The view list specifies the location and type of each field used to display a property value. View lists are described in "View Lists" beginning on page 5-123.

Listing 5-3 A simple information page

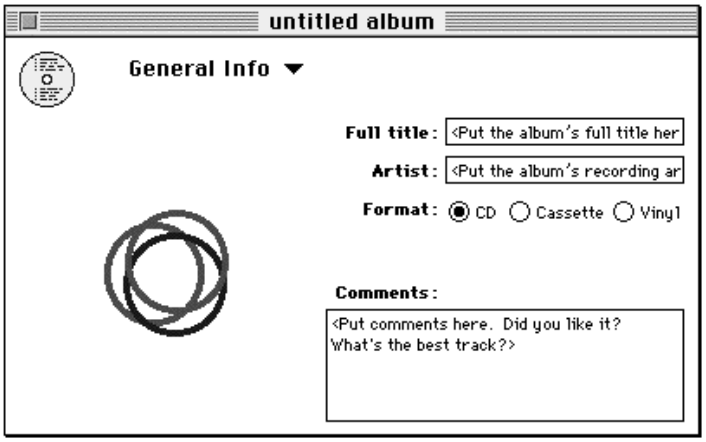
```
#define kAlbumInfoPage  kDETThirdID


resource 'deti' (kAlbumInfoPage, purgeable) {
   1000,                              // sort order
   {0, 0, 0, 0},                      // rectangle to put sublist in
   selectFirstText,                   // select the first text
                                      //  field when info-page opens


   {                                  // the header view list
      kDETNoProperty, kDETNoProperty, kAlbumInfoPage;
   },
   {                                  // no subview view lists
```

```
   }
};

resource 'rstr' (kAlbumInfoPage + kDETTemplateName, purgeable) {
   "WAVE Album 1st Info Page"          // start with application signature
};

resource 'rstr' (kAlbumInfoPage + kDETInfoPageName, purgeable) {
   "General Info"
};

// Associate this information page with records of this type
//  and with the aspect

resource 'rstr' (kAlbumInfoPage + kDETRecordType, purgeable) {
   "WAVE Album"
};

resource 'rstr' (kAlbumInfoPage + kDETInfoPageMainViewAspect, purgeable) {
   "WAVE Album First Info Page Aspect"
};

// View list

#define kCoverTop             (kDETSubpageIconBottom + 8)
#define kCoverLeft            (kDETSubpageIconLeft - 2)
#define kCoverBottom          (kCoverTop + 175)
#define kCoverRight           (kCoverLeft + 175)

#define k1stColumnLeft        (kCoverRight + 4)
#define k1stColumnRight       (k1stColumnLeft + 65)
#define k2ndColumnLeft        (k1stColumnRight + 4)
#define k2ndColumnRight       (kDETRecordInfoWindWidth - 8)

#define kTitleTop             (kCoverTop)
#define kTitleBottom          (kTitleTop + kDETAppFontLineHeight + 4)
#define kArtistTop            (kTitleBottom + 6)
#define kArtistBottom         (kArtistTop + kDETAppFontLineHeight + 4)
#define kFormatTop            (kArtistBottom + 6)
#define kFormatBottom         (kFormatTop + kDETAppFontLineHeight + 4)
#define kNumFormats           (3)
#define kCDRadioLeft          (k2ndColumnLeft)
#define kCDRadioRight         (kCDRadioLeft + 35)
```

```
#define kCassetteRadioLeft    (kCDRadioRight)
#define kCassetteRadioRight   (kCassetteRadioLeft + 60)
#define kVinylRadioLeft       (kCassetteRadioRight)
#define kVinylRadioRight      (k2ndColumnRight)
#define kCommentsTop          (kFormatBottom + 32)
#define kCommentsLabelBottom  (kCommentsTop + kDETAppFontLineHeight + 4)
#define kCommentsBottom       (kCoverBottom)


resource 'detv' (kAlbumInfoPage, purgeable) {
   {
   kDETSubpageIconRect, kDETNoFlags, kDETAspectMainBitmap,
   Bitmap { kDETLargeIcon };


   {kTitleTop, k1stColumnLeft, kTitleBottom, k1stColumnRight},
   kDETNoFlags, kDETNoProperty,
   StaticTextFromView { kDETApplicationFont, kDETApplicationFontSize,
    kDETRight, kDETBold, "Full title:" };

   {kTitleTop - 2, k2ndColumnLeft, kTitleBottom - 2, k2ndColumnRight},
   kDETEnabled, prTitle,
   EditText { kDETApplicationFont, kDETApplicationFontSize, kDETLeft,
    kDETNormal };


   {kArtistTop, k1stColumnLeft, kArtistBottom, k1stColumnRight},
   kDETNoFlags, kDETNoProperty,
   StaticTextFromView { kDETApplicationFont, kDETApplicationFontSize,
    kDETRight, kDETBold, "Artist:" };

   {kArtistTop - 2, k2ndColumnLeft, kArtistBottom - 2, k2ndColumnRight},
   kDETEnabled, prArtist,
   EditText { kDETApplicationFont, kDETApplicationFontSize, kDETLeft,
    kDETNormal };


   {kFormatTop, k1stColumnLeft, kFormatBottom, k1stColumnRight},
   kDETNoFlags, kDETNoProperty,
   StaticTextFromView { kDETApplicationFont, kDETApplicationFontSize,
    kDETRight, kDETBold, "Format:" };
```

```
    {kFormatTop, kCDRadioLeft, kFormatBottom, kCDRadioRight},
    kDETEnabled, prFormat,
    RadioButton { kDETApplicationFont, kDETApplicationFontSize, kDETLeft,
     kDETNormal, "CD", prFormat, 1 };

    {kFormatTop, kCassetteRadioLeft, kFormatBottom, kCassetteRadioRight},
     kDETEnabled, prFormat,
    RadioButton { kDETApplicationFont, kDETApplicationFontSize, kDETLeft,
     kDETNormal, "Cassette", prFormat, 2 };

    {kFormatTop, kVinylRadioLeft, kFormatBottom, kVinylRadioRight},
     kDETEnabled, prFormat,
    RadioButton { kDETApplicationFont, kDETApplicationFontSize, kDETLeft,
     kDETNormal, "Vinyl", prFormat, 3 };

    {kCommentsTop, k1stColumnLeft, kCommentsLabelBottom, k1stColumnRight},
     kDETNoFlags, kDETNoProperty,
    StaticTextFromView { kDETApplicationFont, kDETApplicationFontSize,
     kDETRight, kDETBold, "Comments:" };

    {kCommentsLabelBottom, k1stColumnLeft, kCommentsBottom - 2,
     k2ndColumnRight},
    kDETEnabled + kDETMultiLine, prComments,
    EditText { kDETApplicationFont, kDETApplicationFontSize, kDETLeft,
     kDETNormal };

    { kCoverTop, kCoverLeft, kCoverBottom, kCoverRight },
     kDETNoFlags, prCover,
    EditPicture { 8 };
    }
};
```

Figure 5-19 Simple information page

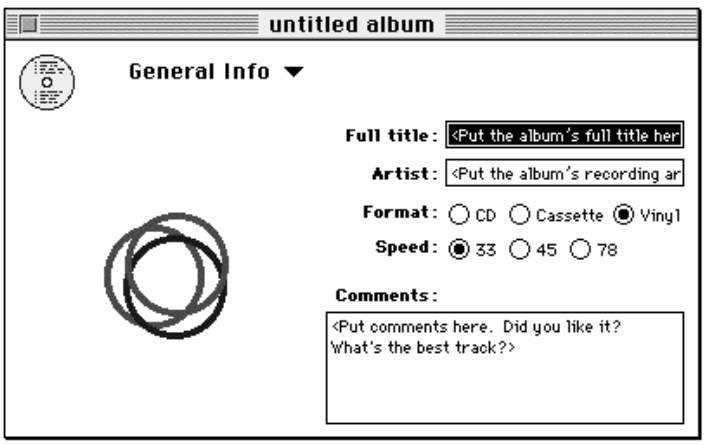


Listing 5-3 together with Listing 5-2 on page 5-34 describe the information page shown in Figure 5-19. The user can type information into the editable text fields and place a figure in the editable picture field.

Adding a Conditional View

A conditional view is one that appears in an information page only if certain conditions are met. For example, the Album information page shown in the preceding example could display radio buttons that specify the speed of the album, but only if the user selects the Vinyl radio button for album format (Figure 5-20).

Figure 5-20 Simple information page with a conditional view



To implement the conditional view shown in Figure 5-20, make the following additions to the templates:

- Add the property `prVinylSpeed` to the list of properties.

```
#define prTitle      kDETFirstDevProperty
#define prArtist     kDETFirstDevProperty + 1
#define prComments   kDETFirstDevProperty + 2
#define prFormat     kDETFirstDevProperty + 3
#define prCover      kDETFirstDevProperty + 4
#define prVinylSpeed kDETFirstDevProperty + 5
```

- Add the `prVinylSpeed` property to the lookup table.

```
resource 'dett' (kAlbum1stInfoPageAspect + kDETAspectLookup,
purgeable) {
   {
   {"WAVE Album General Info"}, typeBinary,
      useForInput, useForOutput, notInSublist, isNotAlias,
       isNotRecordRef,
      {
         'rstr', prTitle, 0;
         'rstr', prArtist, 0;
         'rstr', prComments, 0;
         'word', prFormat, 0;
         'word', prVinylSpeed, 0;
      };
   {"Album Cover"}, typeBinary,
      useForInput, useForOutput, notInSublist, isNotAlias,
       isNotRecordRef,
      { 'rest', prCover , 0 };
   }
};
```

- Add a default property value for the `prVinylSpeed` property.

```
resource 'detn' (kAlbumMainAspect + prVinylSpeed) {
   1
};
```

- Add help balloons.

```
resource 'rst#' (kAlbum1stInfoPageAspect + kDETAspectBalloons,
                 purgeable) {
   {
   "The full title.", "The full title. Uneditable because the
    record is locked or access is restricted.",
```

```
   "The artist or group.", "The artist or group. Uneditable
    because the record is locked or access is restricted.",
   "Comments.", Comments. Uneditable because the record is locked
    or access is restricted.",
   "Format.", "Format. Uneditable because the record is locked or
    access is restricted."
   "Album's cover.", Album's cover. Uneditable because the record
    is locked or access is restricted."
   "Record speed", Record speed. Uneditable because the record is
    locked or access is restricted."
   }
};
```

- Add a line to the information page signature resource for a second view list. Each view list has a corresponding line in the information page signature resource; each line has two property numbers and a resource ID for the view list resource. The view is displayed only if the values of the two properties are equal. In this case, the second line requires that the property `prFormat` must equal 3; that is, the value of the property (`kDETFirstConstantProperty + 3`) is the constant 3. Information page signature resources are defined and described in "Information Page Template Signature Resource" on page 5-121.

```
resource 'deti' (kAlbumInfoPage, purgeable) {
   1000,
   {0, 0, 0, 0},
   selectFirstText,
   {
      kDETNoProperty, kDETNoProperty, kAlbumInfoPage;
      prFormat, kDETFirstConstantProperty + 3, kAlbumInfoPage + 1;
   },
   {
   }
};
```

- Add the definitions and view list for the conditional view. Notice that the conditional view resource (`'detv'`) includes the identification number for the conditional view, `kAlbumInfoPage + 1`.

```
#define kConditionalTop       (kFormatBottom + 4)
#define kConditionalBottom    (kConditionalTop +
                               kDETAppFontLineHeight + 4)
#define k33RadioLeft          (k2ndColumnLeft)
#define k33RadioRight         (kCDRadioLeft + 35)
#define k45RadioLeft          (k33RadioRight)
```

```
#define k45RadioRight         (k45RadioLeft + 35)
#define k78RadioLeft          (k45RadioRight)
#define k78RadioRight         (k45RadioRight + 35)

resource 'detv' (kAlbumInfoPage + 1, purgeable) {
   {
   {kConditionalTop, k1stColumnLeft, kConditionalBottom,
    k1stColumnRight},
   kDETNoFlags, kDETNoProperty,
   StaticTextFromView { kDETApplicationFont,
      kDETApplicationFontSize, kDETRight, kDETBold, "Speed:" };

   {kConditionalTop, k33RadioLeft, kConditionalBottom,
    k33RadioRight},
   kDETEnabled, prVinylSpeed,
   RadioButton { kDETApplicationFont, kDETApplicationFontSize,
      kDETLeft, kDETNormal, "33", prVinylSpeed, 1 };

   {kConditionalTop, k45RadioLeft, kConditionalBottom,
    k45RadioRight}, kDETEnabled, prVinylSpeed,
   RadioButton { kDETApplicationFont, kDETApplicationFontSize,
      kDETLeft, kDETNormal, "45", prVinylSpeed, 2 };

   {kConditionalTop, k78RadioLeft, kConditionalBottom,
    k78RadioRight}, kDETEnabled, prVinylSpeed,
   RadioButton { kDETApplicationFont, kDETApplicationFontSize,
      kDETLeft, kDETNormal, "78", prVinylSpeed, 3 };
   }
};
```

For another example of a conditional view, see “Implementing Conditional Views” beginning on page 5-131.

Adding an Information Page With a Sublist

Listing 5-4 shows the aspect and information page templates for a second information page for the Album record. This information page (shown in Figure 5-21) provides details about the tracks on the album, including a list of all the tracks. The list of tracks is implemented as an information page sublist. Each item in the sublist is an attribute value; each attribute value includes the title and track number of a track on the album. For more information on sublists, see “Sublists” on page 5-136.

Figure 5-21 Information page with a sublist

Note that this template supports the dropping of an attribute into the sublist as a way to add an item to the sublist. The aspect template signature resource, the `kDETAspectDragInString` resource, and the `kDETAspectAttrDragIn` resource all support drops. See "Supporting Drags and Drops" beginning on page 5-98 for more information about resources that support dragging and dropping objects on templates.

Listing 5-4 An information page with a sublist

```
#include "Types.r"
#include "OCETemplates.h"
#include "OCE.r"


// This is an aspect template with this base resource ID.

#define kAlbum2ndInfoPageAspect  kDETFourthID

// Aspect template signature resource

resource 'deta' (kAlbum2ndInfoPageAspect, purgeable) {
   0,                   // drop-operation order
   dropCheckAlways,     // drop-check flag
   notMainAspect        // not the main aspect
};

// Template name

resource 'rstr' (kAlbum2ndInfoPageAspect + kDETTemplateName, purgeable) {
```

```
   "WAVE Album Second Info Page Aspect"
};



// Associate this aspect template with records of type Album.

resource 'rstr' (kAlbum2ndInfoPageAspect + kDETRecordType, purgeable) {
   "WAVE Album"
};


// Icons

include "AlbumIcons" 'ICN#'(0) as
    'ICN#'(kAlbum2ndInfoPageAspect + kDETAspectMainBitmap, purgeable);
include "AlbumIcons" 'icl4'(0) as
    'icl4'(kAlbum2ndInfoPageAspect + kDETAspectMainBitmap, purgeable);
include "AlbumIcons" 'icl8'(0) as
    'icl8'(kAlbum2ndInfoPageAspect + kDETAspectMainBitmap, purgeable);

// Aspect properties - shared between aspect and info page(s)

#define prTrackNumber         kDETFirstDevProperty
#define prNumTracks           kDETFirstDevProperty + 1
#define prPlayingTimeHours    kDETFirstDevProperty + 2
#define prPlayingTimeMinutes  kDETFirstDevProperty + 3
#define prPlayingTimeSeconds  kDETFirstDevProperty + 4


// Lookup table

// This lookup table defines the format of attribute type
//  WAVE Album Track Info. This attribute type is not displayed in a
//  sublist and so does not require a main aspect.
// Attribute values of type WAVE Track are displayed in the sublist.
//  The format of attribute type WAVE Track is defined in the main aspect
//  shown in Listing 5-5 on page 5-52.

resource 'dett' (kAlbum2ndInfoPageAspect + kDETAspectLookup, purgeable) {
   {
   {"WAVE Album Track Info"}, typeBinary,
      useForInput, useForOutput, notInSublist, isNotAlias, isNotRecordRef,
```

```
      {
         'word', prNumTracks, 0;
         'long', prPlayingTimeHours, 0;
         'long', prPlayingTimeMinutes, 0;
         'long', prPlayingTimeSeconds, 0;
      };

   {"WAVE Track"}, typeBinary,
      notForInput, notForOutput, useInSublist, isNotAlias, isNotRecordRef,
      {};
   }
};

// Drag and drop information (see
//  “Supporting Drags and Drops” beginning on page 5-98)

// Prompt string for drag-in dialog box
resource 'rstr' (kAlbum2ndInfoPageAspect + kDETAspectDragInString,
                  purgeable) {
   "Do you want to add %3%“^3”%the selected items% to the track address list
of *0x/the/* ^1 “^2”?"
};

// Attributes can be dragged from any kind of record (""); attributes of
//  type WAVE Track can be dragged into this record; and the new copy of the
//  attribute will be of type WAVE Track.
resource 'rst#' (kAlbum2ndInfoPageAspect + kDETAspectAttrDragIn, purgeable) {
   {
   "", "WAVE Track", "WAVE Track"
   }
};


// Sublist sorting information

// Property names in this resource appear in the View menu, and property
//  numbers tell the CE what to sort by. Positive property number is
//  alphanumeric sort; negative number is numeric sort.

resource 'detm' (kAlbum2ndInfoPageAspect + kDETAspectViewMenu, purgeable) {
   kAlbum2ndInfoPageAspect + kDETAspectViewMenu,
   {
   kDETAspectName,"by Title";
```

```
   -prTrackNumber,"by Track Number";
   }
};

// Properties in this resource are sorted in reverse order.

resource 'detp' (kAlbum2ndInfoPageAspect + kDETAspectReverseSort, purgeable)
{
   {
   prTrackNumber
   }
};

// Text for help balloons for the properties

resource 'rst#' (kAlbum2ndInfoPageAspect + kDETAspectBalloons,purgeable) {
   {
   "The number of tracks on the album.", The number of tracks on the album.
    Uneditable because the record is locked or access is restricted.",
   "The number of hours of music on the album.", "The number of hours of
    music on the album. Uneditable because the record is locked or access
     is restricted.",
   "The number of minutes of music on the album.", "The number of minutes of
    music on the album. Uneditable because the record is locked or access
     is restricted.",
   "The number of seconds of music on the album.", "The number of seconds of
    music on the album. Uneditable because the record is locked or access
     is restricted.",
   }
};
// ---------------------------------------------------------------------
//
// Album information page



#define kAlbum2ndInfoPage     kDETFifthID

#define kTitleTop             (85)
#define kTitleBottom          (kTitleTop + 12)
#define kSublistTop           (kTitleBottom + 2)
#define kSublistBottom        (kDETRecordInfoWindHeight - 40)
#define kSublistLeft          (12)
```

```
#define kSublistRight         (kDETRecordInfoWindWidth - 12)

// Information page template signature resource

resource 'deti' (kAlbum2ndInfoPage, purgeable) {
   2000,
   {kSublistTop, kSublistLeft, kSublistBottom, kSublistRight},
   selectFirstText,
// View list for main view is identified by the following line.
   {
      kDETNoProperty, kDETNoProperty, kAlbum2ndInfoPage;
   },
// View list for sublist is identified by this line.
   {
      kDETNoProperty, kDETNoProperty, kAlbum2ndInfoPage + 1;
   }
};

resource 'rstr' (kAlbum2ndInfoPage + kDETTemplateName, purgeable) {
   "WAVE Album 2nd Info Page"
};

resource 'rstr' (kAlbum2ndInfoPage + kDETInfoPageName, purgeable) {
   "Track Info"
};

// Associate this information page with records of type WAVE Album
//  and with this aspect template.

resource 'rstr' (kAlbum2ndInfoPage + kDETRecordType, purgeable) {
   "WAVE Album"
};

resource 'rstr' (kAlbum2ndInfoPage + kDETInfoPageMainViewAspect, purgeable) {
   "WAVE Album Second Info Page Aspect"
};


// View list - what you see in this information page

#define kMyFirstColumnLeft    (55)
#define kMyFirstColumnRight   (kMyFirstColumnLeft + 120)
#define kEditTextWidth        (23)
```

```
#define kSpaceBeforeEditDesc  (25)
#define kNumEditColumns       (3)
#define kMyEditColumnWidth    (70)
#define k1stEditColumnLeft    (kMyFirstColumnRight + 2)
#define k2ndEditColumnLeft    (k1stEditColumnLeft + kMyEditColumnWidth)
#define k3rdEditColumnLeft    (k2ndEditColumnLeft + kMyEditColumnWidth)
#define k4thEditColumnLeft    (k3rdEditColumnLeft + kMyEditColumnWidth)

#define kNumTracksTop         (40)
#define kNumTracksBottom      (kNumTracksTop + kDETAppFontLineHeight + 4)
#define kPlayingTimeTop       (kNumTracksBottom + 4)
#define kPlayingTimeBottom    (kPlayingTimeTop + kDETAppFontLineHeight + 4)
#define k2ndColumnRightInset  (kDETRecordInfoWindWidth - 10)

#define kButtonTop            (kSublistBottom + 15)
#define kButtonBottom         (kButtonTop + 16)
#define kOpenLeft             62
#define kOpenRight            112
#define kAddLeft              208
#define kAddRight             258
#define kRemoveLeft           270
#define kRemoveRight          320

#define kIconLeft             2
#define kNameLeft             22
#define kTrackNumberLeft      162
#define kPrefLeft             285
#define kPrefRight            305

#define kIconEntryTop         -7
#define kIconEntryBottom      9
#define kEntryTop             -5
#define kEntryBottom          9

resource 'detv' (kAlbum2ndInfoPage, purgeable) {
   {
   kDETSubpageIconRect, kDETNoFlags, kDETAspectMainBitmap,
   Bitmap { kDETLargeIcon };

   {kNumTracksTop, kMyFirstColumnLeft, kNumTracksBottom,
    kMyFirstColumnRight},
      kDETNoFlags, kDETNoProperty,
      StaticTextFromView { kDETApplicationFont, kDETApplicationFontSize,
```

```
    kDETRight, kDETBold, "Number of tracks:" };

{kNumTracksTop - 2, k1stEditColumnLeft, kNumTracksBottom - 2,
 k1stEditColumnLeft + kEditTextWidth},
   kDETEnabled + kDETNumericOnly, prNumTracks,
   EditText { kDETApplicationFont, kDETApplicationFontSize, kDETLeft,
    kDETNormal };

{kPlayingTimeTop, kMyFirstColumnLeft, kPlayingTimeBottom,
 kMyFirstColumnRight},
   kDETNoFlags, kDETNoProperty,
   StaticTextFromView { kDETApplicationFont, kDETApplicationFontSize,
    kDETRight, kDETBold, "Total playing time:" };

{kPlayingTimeTop - 2, k1stEditColumnLeft, kPlayingTimeBottom - 2,
 k1stEditColumnLeft + kEditTextWidth},
   kDETEnabled + kDETNumericOnly, prPlayingTimeHours,
   EditText { kDETApplicationFont, kDETApplicationFontSize, kDETLeft,
    kDETNormal };

{kPlayingTimeTop, k1stEditColumnLeft + kSpaceBeforeEditDesc,
 kPlayingTimeBottom, k2ndEditColumnLeft},
   kDETNoFlags, kDETNoProperty,
   StaticTextFromView { kDETApplicationFont, kDETApplicationFontSize,
    kDETLeft, kDETNormal, "Hours" };

{kPlayingTimeTop - 2, k2ndEditColumnLeft, kPlayingTimeBottom - 2,
 k2ndEditColumnLeft + kEditTextWidth},
   kDETEnabled + kDETNumericOnly, prPlayingTimeMinutes,
   EditText { kDETApplicationFont, kDETApplicationFontSize, kDETLeft,
    kDETNormal };

{kPlayingTimeTop, k2ndEditColumnLeft + kSpaceBeforeEditDesc,
 kPlayingTimeBottom, k3rdEditColumnLeft},
   kDETNoFlags, kDETNoProperty,
   StaticTextFromView { kDETApplicationFont, kDETApplicationFontSize,
    kDETLeft, kDETNormal, "Minutes" };

{kPlayingTimeTop - 2, k3rdEditColumnLeft, kPlayingTimeBottom - 2,
 k3rdEditColumnLeft + kEditTextWidth},
   kDETEnabled + kDETNumericOnly, prPlayingTimeSeconds,
   EditText { kDETApplicationFont, kDETApplicationFontSize, kDETLeft,
    kDETNormal };
```

```
   {kPlayingTimeTop, k3rdEditColumnLeft + kSpaceBeforeEditDesc,
    kPlayingTimeBottom, k4thEditColumnLeft},
      kDETNoFlags, kDETNoProperty,
      StaticTextFromView { kDETApplicationFont, kDETApplicationFontSize,
       kDETLeft, kDETNormal, "Seconds" };

   {kSublistTop - 1, kSublistLeft - 1, kSublistBottom + 1,
    kSublistRight + 1},
      kDETNoFlags, kDETNoProperty,
      Box { kDETUnused };

   {kTitleTop, kSublistLeft + kNameLeft, kTitleBottom,
    kSublistLeft + kTrackNumberLeft - 2},
      kDETNoFlags, kDETAspectName,
      StaticCommandTextFromView { kDETDefaultFont, kDETDefaultFontSize,
       kDETLeft, kDETUnderline, "Title", kDETChangeViewCommand,  - 1};

   {kTitleTop, kSublistLeft + kTrackNumberLeft, kTitleBottom,
    kSublistLeft + kPrefLeft - 2},
      kDETNoFlags, prTrackNumber,
      StaticCommandTextFromView { kDETDefaultFont, kDETDefaultFontSize,
       kDETLeft, kDETNormal, "Track Number", kDETChangeViewCommand, - 2 };

   {kButtonTop, kOpenLeft, kButtonBottom, kOpenRight},
      kDETNoFlags, kDETOpenSelectedItems,
      Button { kDETApplicationFont, 10, kDETCenter, kDETNormal, "Open",
       kDETOpenSelectedItems };

   {kButtonTop, kAddLeft, kButtonBottom, kAddRight},
      kDETNoFlags, kDETAddNewItem,
      Button { kDETApplicationFont, 10, kDETCenter, kDETNormal, "Add…",
       kDETAddNewItem };

   {kButtonTop, kRemoveLeft, kButtonBottom, kRemoveRight},
      kDETNoFlags, kDETRemoveSelectedItems,
      Button { kDETApplicationFont, 10, kDETCenter, kDETNormal, "Remove",
       kDETRemoveSelectedItems };
   }
};

// View list for sublist
```

```
resource 'detv' (kAlbum2ndInfoPage + 1, purgeable) {
   {
   {kIconEntryTop, kIconLeft, kIconEntryBottom, kNameLeft-4},
      kDETHilightIfSelected, kDETAspectMainBitmap,
      Bitmap { kDETMiniIcon };

   {kEntryTop, kNameLeft, kEntryBottom, kTrackNumberLeft - 2},
      kDETHilightIfSelected + kDETDynamicSize, kDETAspectName,
      EditText { kDETDefaultFont, kDETDefaultFontSize, kDETLeft,
       kDETNormal };

   {kEntryTop, kTrackNumberLeft, kEntryBottom, kPrefLeft - 2},
      kDETHilightIfSelected + kDETDynamicSize, prTrackNumber,
      EditText { kDETDefaultFont, kDETDefaultFontSize, kDETLeft,
       kDETNormal };
   }
};
```

Writing a Main Aspect and Information Page for an Attribute

The information page in Listing 5-4 on page 5-44 allows a user to add a new attribute of type Track. To make this possible, you have to provide a main aspect for attributes of that type. To let the user see the contents of the attribute, you need to provide an information page (see Figure 5-5 on page 5-9). Listing 5-5 shows the main aspect template and information page template for attributes of type Track. Because this is an attribute, the main aspect template contains all the properties needed by the information page in addition to the resources required for a main aspect template.

Listing 5-5 Attribute main aspect and information page

```
#include "Types.r"
#include "OCETemplates.h"
#include "OCE.r"
#include "Track.h"

#define kDETSixthID     (1000 + 5 * kDETIDSep)
#define kTrackAspect    (kDETSixthID + kDETIDSep)
#define kTrackInfoPage  (kDETSixthID + (2 * kDETIDSep))


// The aspect template

resource 'deta' (kTrackAspect, purgeable) {
```

```
   0,                // drop-operation order
   dropCheckAlways,  // drop-check flag
   isMainAspect      // is the main aspect
};

resource 'rstr' (kTrackAspect + kDETTemplateName, purgeable) {
   "WAVE Track Aspect"
};

resource 'rstr' (kTrackAspect + kDETAttributeType, purgeable) {
   "WAVE Track"
};

resource 'rstr' (kTrackAspect + kDETAspectKind, purgeable) {
   "Track"
};

resource 'rstr' (kTrackAspect + kDETAspectWhatIs, purgeable) {
   "Track\n\nA track on an album."
};

resource 'rst#' (kTrackAspect + kDETAspectCategory, purgeable)
   {{
   "Recordings"
   }};

resource 'rstr' (kTrackAspect + kDETAspectNewMenuName, purgeable) {
   "New Track"
};

#define prTrackNumber      kDETFirstDevProperty
#define prTrackMinutes     (kDETFirstDevProperty + 1)
#define prTrackSeconds     (kDETFirstDevProperty + 2)
#define prTrackComposer    (kDETFirstDevProperty + 3)
#define prTrackComments    (kDETFirstDevProperty + 4)

// Default values for a newly created attribute

data 'detb' (kTrackAspect + kDETAspectNewValue, purgeable) {
   $"626E 7279"                          // tag (bnry)
   $"0000 0001"                          // prTrackNumber (1)
   $"0000 0000"                          // prTrackMinutes (1)
   $"0000 0000"                          // prTrackSeconds (1)
```

```
   $"0000 0007 3C74 6974 6C65 3E"         // kDETAspectName (<title>)
   $"0000 000A 3C63 6F6D 706F 7365 723E"  // composer (<composer>)
   $"0000 000A 3C63 6F6D 6D65 6E74 733E"  // comments (<comments>)
};

// Lookup table

resource 'dett' (kTrackAspect + kDETAspectLookup, purgeable) {
   {
   {"WAVE Track"}, typeBinary,
      useForInput, useForOutput, notInSublist, isNotAlias, isNotRecordRef,
      {
      'long', prTrackNumber, 0;
      'long', prTrackMinutes, 0;
      'long', prTrackSeconds, 0;
      'rstr', kDETAspectName, 0;
      'rstr', prTrackComposer, 0;
      'rstr', prTrackComments, 0
      };
   }
};

// Icons
include "TrackIcons" 'ICN#'(0) as 'ICN#'(kTrackAspect + kDETAspectMainBitmap,
       purgeable);
include "TrackIcons" 'icl4'(0) as 'icl4'(kTrackAspect + kDETAspectMainBitmap,
       purgeable);
include "TrackIcons" 'icl8'(0) as 'icl8'(kTrackAspect + kDETAspectMainBitmap,
       purgeable);
include "TrackIcons" 'ics#'(0) as 'ics#'(kTrackAspect + kDETAspectMainBitmap,
       purgeable);
include "TrackIcons" 'ics4'(0) as 'ics4'(kTrackAspect + kDETAspectMainBitmap,
       purgeable);
include "TrackIcons" 'ics8'(0) as 'ics8'(kTrackAspect + kDETAspectMainBitmap,
       purgeable);
include "TrackIcons" 'SICN'(0) as 'SICN'(kTrackAspect + kDETAspectMainBitmap,
       purgeable);

// ----------------------------------------------------------------------

// Information page

#define kTrackNumberTop          (50)
```

```
#define kTrackNumberBottom       (kTrackNumberTop + kDETAppFontLineHeight
                                  + 4)
#define kTrackPlayingTimeTop     (kTrackNumberBottom + 4)
#define kTrackPlayingTimeBottom  (kTrackPlayingTimeTop +
                                  kDETAppFontLineHeight + 4)
#define kTrackComposerTop        (kTrackPlayingTimeBottom + 4)
#define kTrackComposerBottom     (kTrackComposerTop +
                                  kDETAppFontLineHeight + 4 +
                                  kDETAppFontLineHeight + 4 +
                                  kDETAppFontLineHeight + 4)
#define kTrackCommentsTop        (kTrackComposerBottom + 4)
#define kTrackCommentsBottom     (kTrackCommentsTop +
                                  kDETAppFontLineHeight + 4 +
                                  kDETAppFontLineHeight + 4 +
                                  kDETAppFontLineHeight + 4 +
                                  kDETAppFontLineHeight + 4 +
                                  kDETAppFontLineHeight + 4)

#define kTrackEditTextWidth      (23)
#define kTrackSpaceBeforeEditDesc(25)
#define kTrack1stColumnLeft      (4)
#define kTrack1stColumnRight     (kDETAttributeInfoWindWidth / 2 - 20)
#define kTrack2ndColumnLeft      (kTrack1stColumnRight + 4)
#define kTrack2ndColumnRight     (kDETAttributeInfoWindWidth - 8)
#define kTrackSecondsColumnLeft  (kTrack2ndColumnLeft +
                                  kTrackSpaceBeforeEditDesc + 40)

resource 'deti' (kTrackInfoPage, purgeable) {
   1000,
   {0, 0, 0, 0},
   selectFirstText,
   {
   kDETNoProperty, kDETNoProperty, kTrackInfoPage;
   },
   {
   }
};

resource 'rstr' (kTrackInfoPage + kDETTemplateName, purgeable) {
   "WAVE Track Info Page"
};

resource 'rstr' (kTrackInfoPage + kDETAttributeType, purgeable) {
```

```
   "WAVE Track"
};

resource 'rstr' (kTrackInfoPage + kDETInfoPageName, purgeable) {
   "Track Info"
};

resource 'rstr' (kTrackInfoPage + kDETInfoPageMainViewAspect, purgeable) {
   "WAVE Track Aspect"
};

// View list

resource 'detv' (kTrackInfoPage, purgeable) {
   {
   kDETSubpageIconRect, kDETNoFlags, kDETAspectMainBitmap,
   Bitmap { kDETLargeIcon };

   {kTrackNumberTop, kTrack1stColumnLeft, kTrackNumberBottom,
       kTrack1stColumnRight},
      kDETNoFlags, kDETNoProperty,
      StaticTextFromView { kDETApplicationFont, kDETApplicationFontSize,
       kDETRight, kDETBold, "Track Number:" };

   {kTrackNumberTop - 2, kTrack2ndColumnLeft, kTrackNumberBottom - 2,
       kTrack2ndColumnLeft + kTrackEditTextWidth},
      kDETEnabled + kDETNumericOnly, prTrackNumber,
      EditText { kDETApplicationFont, kDETApplicationFontSize, kDETLeft,
       kDETNormal };

   {kTrackPlayingTimeTop, kTrack1stColumnLeft, kTrackPlayingTimeBottom,
       kTrack1stColumnRight},
      kDETNoFlags, kDETNoProperty,
      StaticTextFromView { kDETApplicationFont, kDETApplicationFontSize,
       kDETRight, kDETBold, "Playing Time:" };

   {kTrackPlayingTimeTop - 2, kTrack2ndColumnLeft,
       kTrackPlayingTimeBottom - 2,
       kTrack2ndColumnLeft + kTrackEditTextWidth},
      kDETEnabled + kDETNumericOnly, prTrackMinutes,
      EditText { kDETApplicationFont, kDETApplicationFontSize, kDETLeft,
       kDETNormal };
```

```
{kTrackPlayingTimeTop, kTrack2ndColumnLeft + kTrackSpaceBeforeEditDesc,
    kTrackPlayingTimeBottom, kTrackSecondsColumnLeft},
   kDETNoFlags, kDETNoProperty,
StaticTextFromView { kDETApplicationFont, kDETApplicationFontSize,
    kDETLeft, kDETNormal, "Mins" };

{kTrackPlayingTimeTop - 2, kTrackSecondsColumnLeft,
    kTrackPlayingTimeBottom - 2, kTrackSecondsColumnLeft +
    kTrackEditTextWidth},
   kDETEnabled + kDETNumericOnly, prTrackSeconds,
   EditText { kDETApplicationFont, kDETApplicationFontSize, kDETLeft,
    kDETNormal };

{kTrackPlayingTimeTop, kTrackSecondsColumnLeft +
    kTrackSpaceBeforeEditDesc, kTrackPlayingTimeBottom,
    kTrack2ndColumnRight},
   kDETNoFlags, kDETNoProperty,
   StaticTextFromView { kDETApplicationFont, kDETApplicationFontSize,
    kDETLeft, kDETNormal,"Secs" };

{kTrackComposerTop, kTrack1stColumnLeft, kTrackComposerBottom,
    kTrack1stColumnRight},
   kDETNoFlags, kDETNoProperty,
StaticTextFromView { kDETApplicationFont, kDETApplicationFontSize,
    kDETRight, kDETBold,"Composer:" };

{kTrackComposerTop - 2, kTrack2ndColumnLeft, kTrackComposerBottom - 2,
    kTrack2ndColumnRight},
   kDETEnabled + kDETMultiLine, prTrackComposer,
   EditText { kDETApplicationFont, kDETApplicationFontSize, kDETLeft,
    kDETNormal };

{kTrackCommentsTop, kTrack1stColumnLeft, kTrackCommentsBottom,
    kTrack1stColumnRight},
   kDETNoFlags, kDETNoProperty,
StaticTextFromView { kDETApplicationFont, kDETApplicationFontSize,
    kDETRight, kDETBold,"Comments:" };

{kTrackCommentsTop - 2, kTrack2ndColumnLeft, kTrackCommentsBottom - 2,
    kTrack2ndColumnRight},
   kDETEnabled + kDETMultiLine, prTrackComments,
   EditText { kDETApplicationFont, kDETApplicationFontSize, kDETLeft,
```

```
        kDETNormal };
    }
};
```

Creating a Custom Information Page Window

The aspect and information page templates in Listing 5-6 define a new AOCE record type and the information page that displays the record's contents. The user can use this record type to store information about collections of recording albums. The information page lists the albums in the collection. Because an AOCE record cannot contain another record, the AOCE record type Album Collection actually contains aliases to records of type Album.

The information page consists of a sublist listing the albums in the collection. The information page window is a custom size, defined by the `'detw'` resource with a resource ID of `kCollectionAspect + kDETAspectInfoPageCustomWindow` (see page 5-97 for a description of this resource). This resource specifies the flag `discludePopup`, so the Catalogs Extension does not include a pop-up menu in the window. The template does not add a custom pop-up menu either. Therefore, no one can add any more information pages to this information page window, because the user would have no way of selecting which page to look at. For this reason, a single aspect template is used for both the main aspect and the information page aspect for this new record type.

Notice also that the aspect template for the Album Collection record type includes a view list. Ordinarily, you can put view lists only in information page templates, not in aspect templates. However, because this aspect template defines a custom information page window, you can include a view list with a resource ID of `kCollectionAspect + kDETAspectInfoPageCustomWindow`. The views defined by this view list appear in every information page associated with this main aspect. (In Listing 5-6, there's only one information page, so the view list could be placed in either the aspect or information page template.)

Listing 5-6 Templates for a custom information page

```
#include "Types.r"
#include "OCETemplates.h"
#include "OCE.r"

#define kCollectionAspect     (kDETFifthID + (3 * kDETIDSep))
#define kCollectionInfoPage   (kDETFifthID + (4 * kDETIDSep))

#define kGeneva 3
#define kSystemFont 0

// Page layout defines
```

```
#define iconColumnWidth       16
#define nameColumnWidth       132
#define kindColumnWidth       86
#define spaceBetweenColumns   2

#define iconColumnLeft        0
#define iconColumnRight       (iconColumnLeft + iconColumnWidth)
#define nameColumnLeft        (iconColumnRight + spaceBetweenColumns)
#define nameColumnRight       (nameColumnLeft + nameColumnWidth)
#define kindColumnLeft        nameColumnRight
#define kindColumnRight       (kindColumnLeft + kindColumnWidth)

#define sublistIconTop        (-7)
#define sublistIconBottom     (9)
#define sublistTextTop        (-6)
#define sublistTextBottom     (8)
#define sublistTitleTop       (35)
#define sublistTitleBottom    (sublistTitleTop + 12)

#define windowHeight          280
#define windowWidth           (kindColumnLeft + kindColumnWidth + 15 + 16)

#define sublistTopBound       (sublistTitleBottom + 2)
#define sublistBottomBound    (windowHeight - 12)
#define sublistLeftBound      12
#define sublistRightBound     (windowWidth - 12)

#define kPageBitmapLeft       (11)
#define kPageBitmapRight      (kPageBitmapLeft + 16)
#define kPageBitmapTop        7
#define kPageBitmapBottom     23
```

```
// Aspect template; serves as both main aspect and information page aspect.
//
resource 'deta' (kCollectionAspect, purgeable) {
   0,                   // drop-operation order
   dropCheckConflicts,  // drop-check flag
   isMainAspect         // is the main aspect
};
```

```
resource 'rstr' (kCollectionAspect + kDETTemplateName, purgeable) {
   "WAVE Album Collection Aspect"
};

resource 'rstr' (kCollectionAspect + kDETRecordType, purgeable) {
   "WAVE Album Collection"
};

resource 'rstr' (kCollectionAspect + kDETAspectKind, purgeable) {
   "album collection"
};

resource 'rstr' (kCollectionAspect + kDETAspectWhatIs, purgeable) {
   "Album Collection\n\nA collection of albums.Open this icon to display
    information about the collection."
};

resource 'rstr' (kCollectionAspect + kDETAspectAliasKind, purgeable) {
   "album collection alias"
};

resource 'rstr' (kCollectionAspect + kDETAspectAliasWhatIs, purgeable) {
   "Album Collection alias\n\This is an alias to a collection of albums.
    Open this alias to display information about the collection."
};

// Record category; this record type is assigned to the same category as the
//  Album record type.

resource 'rst#' (kCollectionAspect + kDETAspectCategory,purgeable)
   {{
   "Recordings"
   }};

// Define a custom information page window.

resource 'detw' (kCollectionAspect + kDETAspectInfoPageCustomWindow,
                  purgeable) {
   { 0, 0, windowHeight, windowWidth },
   discludePopup
};

// View list for views to appear in all information pages for this
```

```
//  main aspect

resource 'detv' (kCollectionAspect + kDETAspectInfoPageCustomWindow,
                  purgeable)
   {
   {
   {6, kPageBitmapRight + 8, 25, kPageBitmapRight + 8 + 166},
    kDETNoFlags, kDETInfoPageNumber,
      Menu {kSystemFont, 12, kDETLeft, kDETNormal, "", kDETInfoPageNumber,
            kDETInfoPageNumber };
   };
   };

resource 'rstr' (kCollectionAspect + kDETAspectNewMenuName, purgeable) {
   "New Album Collection"
};

resource 'rstr' (kCollectionAspect + kDETAspectNewEntryName, purgeable) {
   "untitled album collection"
};

include "AlbumCollectionIcons" 'ICN#'(0) as 'ICN#'(kCollectionAspect +
kDETAspectMainBitmap, purgeable);
include "AlbumCollectionIcons" 'icl4'(0) as 'icl4'(kCollectionAspect +
kDETAspectMainBitmap, purgeable);
include "AlbumCollectionIcons" 'icl8'(0) as 'icl8'(kCollectionAspect +
kDETAspectMainBitmap, purgeable);
include "AlbumCollectionIcons" 'ics#'(0) as 'ics#'(kCollectionAspect +
kDETAspectMainBitmap, purgeable);
include "AlbumCollectionIcons" 'ics4'(0) as 'ics4'(kCollectionAspect +
kDETAspectMainBitmap, purgeable);
include "AlbumCollectionIcons" 'ics8'(0) as 'ics8'(kCollectionAspect +
kDETAspectMainBitmap, purgeable);
include "AlbumCollectionIcons" 'SICN'(0) as 'SICN'(kCollectionAspect +
kDETAspectMainBitmap, purgeable);

// Supporting drops

resource 'rstr' (kCollectionAspect + kDETAspectDragInString, purgeable) {
   "Do you want to add %3%“^3”%the selected items% to *0x/the/* ^1 “^2”?"
};

resource 'rst#' (kCollectionAspect + kDETAspectRecordCatDragIn,purgeable)
```

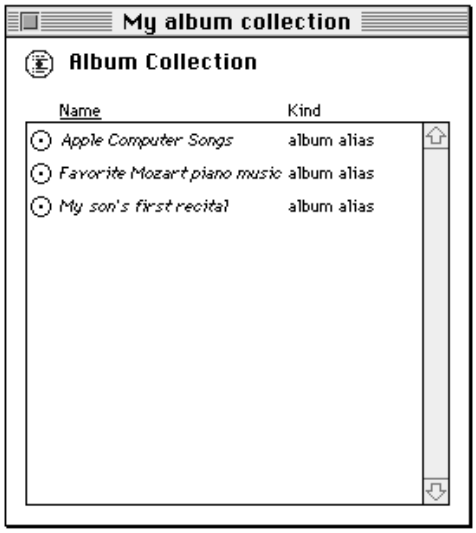
```
   {{
   "Recordings", kMemberAttrTypeBody
   }};

resource 'rst#' (kCollectionAspect + kDETAspectAttrDragIn,purgeable)
   {{
   "", kMemberAttrTypeBody, kMemberAttrTypeBody
   }};

resource 'dett' (kCollectionAspect + kDETAspectLookup, purgeable)
   {{
   {kMemberAttrTypeBody}, typePackedDSSpec,
      notForInput, notForOutput, useInSublist, isAlias, isNotRecordRef,
      {};
   }};

resource 'detm' (kCollectionAspect + kDETAspectViewMenu, purgeable)
   {
   kCollectionAspect + kDETAspectViewMenu,
   {
   kDETPrName, "by Name";
   kDETPrKind, "by Kind";
   }
   };


//-----------------------------------------------------------------------

// The information page template

#define k2ndColumnRightInset  (kDETRecordInfoWindWidth-kDETSubpageRightInset)


resource 'deti' (kCollectionInfoPage, purgeable) {
   1000,
   {sublistTopBound, sublistLeftBound, sublistBottomBound,
    sublistRightBound},
   noSelectFirstText,
   {
   kDETNoProperty, kDETNoProperty, kCollectionInfoPage;
   },
   {
   kDETNoProperty, kDETNoProperty, kCollectionInfoPage + 1;
```

```
   }};

resource 'rstr' (kCollectionInfoPage + kDETTemplateName, purgeable) {
   "WAVE Album Collection Info Page"
};

resource 'rstr' (kCollectionInfoPage + kDETRecordType, purgeable) {
   "WAVE Album Collection"
};

resource 'rstr' (kCollectionInfoPage + kDETInfoPageName, purgeable) {
   "Album Collection"
};

resource 'rstr' (kCollectionInfoPage + kDETInfoPageMainViewAspect,
purgeable) {
   "WAVE Album Collection Aspect"
};

resource 'detv' (kCollectionInfoPage, purgeable)
   {
   {
   {kPageBitmapTop, kPageBitmapLeft, kPageBitmapBottom, kPageBitmapRight},
      kDETNoFlags, kDETAspectMainBitmap,
      Bitmap { kDETSmallIcon };

   {sublistTopBound - 1, sublistLeftBound - 1, sublistBottomBound + 1,
      sublistRightBound + 1}, kDETNoFlags, kDETNoProperty,
      Box { kDETUnused };

   {sublistTitleTop, sublistLeftBound + nameColumnLeft, sublistTitleBottom,
    sublistLeftBound + nameColumnRight},
      kDETNoFlags, kDETPrName,
      StaticCommandTextFromView { kGeneva, 9, kDETLeft, kDETUnderline,
       "Name", kDETChangeViewCommand, - 1 };

   {sublistTitleTop, sublistLeftBound + kindColumnLeft, sublistTitleBottom,
    sublistLeftBound + kindColumnRight},
      kDETNoFlags, kDETPrKind,
      StaticCommandTextFromView { kGeneva, 9, kDETLeft, kDETNormal, "Kind",
       kDETChangeViewCommand, - 2 };
   };
   };
```

```
resource 'detv' (kCollectionInfoPage + 1, purgeable)
   {
   {
   {sublistIconTop, iconColumnLeft, sublistIconBottom, iconColumnRight},
      kDETHilightIfSelected, kDETAspectMainBitmap,
      Bitmap { kDETMiniIcon };

   {sublistTextTop, nameColumnLeft, sublistTextBottom, nameColumnRight},
      kDETHilightIfSelected + kDETDynamicSize, kDETPrName,
      StaticText { kGeneva, 9, kDETLeft, kDETIconStyle };

   {sublistTextTop, kindColumnLeft, sublistTextBottom, kindColumnRight},
      kDETNoFlags, kDETPrKind,
      StaticText { kGeneva, 9, kDETLeft, kDETNormal };
   }
   };
```

Figure 5-22 shows an example of the information page defined by Listing 5-6. Notice that this information page contains no Add or Remove buttons. The only way for a user to add a record alias to a record of type Album Collection is to drag an Album record into the sublist. The only way to remove one is to drag it from the sublist into the Trash. This design works well for the Album Collection record type because letting the user create a new, empty attribute for this record would make little sense. When the user double-clicks an album in the sublist, the Catalogs Extension opens the information page for the album, not for an attribute in the Album record.

Figure 5-22 Custom information page



Writing Template Code Resources

The set of templates you've seen so far creates information pages that let a user store information about an album, about each track on an album, and about a collection of albums. Because the user can enter the duration of each track in the Track Info attribute information page (Figure 5-5 on page 5-9), you can provide a code resource that automatically adds up the number of tracks and the total playing time so that the user does not have to enter that information into editable text boxes. The resulting information page (Figure 5-23) is identical to an earlier information page (Figure 5-21 on page 5-44) except that the number of tracks and total playing time are no longer editable text.

Figure 5-23 Information page using a code resource

The aspect and information page templates that create the information page in Figure 5-23 are identical to those in Listing 5-4 on page 5-44, with the following exceptions:

- The aspect template includes the code resource. To include the code shown in Listing 5-8 on page 5-68 (assuming this code has been compiled and saved as the resource `Album2Code` of type `'detc'` with a resource ID of 0), add the following line to the aspect template:

```
include "Album2Code" 'detc'(0) as
    'detc'(kAlbum2ndInfoPageAspect + kDETAspectCode, purgeable);
```

- The lookup table does not contain the attribute Album Track Info or elements for the properties `prNumTracks`, `prPlayingTimeHours`, `prPlayingTimeMinutes`, or `prPlayingTimeSeconds`. These properties are all handled by the code resource.
- Instead of the edit-text views in the view lists for the "Number of tracks" and "Playing time" fields, the view list contains static text fields that get the values to display from the code resource. Listing 5-7 shows the view lists.

Listing 5-7 View lists that get values from a code resource

```
resource 'detv' (kAlbum2ndInfoPage + 1, purgeable) {
   {
   {kNumTracksTop, kMyFirstColumnLeft, kNumTracksBottom,
    kMyFirstColumnRight},
   kDETNoFlags, kDETNoProperty,
   StaticTextFromView { kDETApplicationFont,
    kDETApplicationFontSize, kDETRight, kDETBold, "Number of
    tracks:" };

   {kNumTracksTop, k1stEditColumnLeft, kNumTracksBottom,
    k1stEditColumnLeft + kEditTextWidth},
   kDETNoFlags, prNumTracks,
   StaticText { kDETApplicationFont, kDETApplicationFontSize,
    kDETLeft, kDETNormal };
   }
};

resource 'detv' (kAlbum2ndInfoPage + 2, purgeable) {
   {
   {kPlayingTimeTop, kMyFirstColumnLeft, kPlayingTimeBottom,
    kMyFirstColumnRight},
   kDETNoFlags, kDETNoProperty,
   StaticTextFromView { kDETApplicationFont,
    kDETApplicationFontSize, kDETRight, kDETBold,
    "Total playing time:" };
```

```
    {kPlayingTimeTop, k1stEditColumnLeft, kPlayingTimeBottom,
     k1stEditColumnLeft + kEditTextWidth},
    kDETNoFlags, prPlayingTimeHours,
    StaticText { kDETApplicationFont, kDETApplicationFontSize,
     kDETLeft, kDETNormal };

    {kPlayingTimeTop, k1stEditColumnLeft + kSpaceBeforeEditDesc,
     kPlayingTimeBottom, k2ndEditColumnLeft},
    kDETNoFlags, kDETNoProperty,
    StaticTextFromView { kDETApplicationFont,
     kDETApplicationFontSize, kDETLeft, kDETNormal, "Hours" };

    {kPlayingTimeTop, k2ndEditColumnLeft, kPlayingTimeBottom,
     k2ndEditColumnLeft + kEditTextWidth},
    kDETNoFlags, prPlayingTimeMinutes,
    StaticText { kDETApplicationFont, kDETApplicationFontSize,
     kDETLeft, kDETNormal };

    {kPlayingTimeTop, k2ndEditColumnLeft + kSpaceBeforeEditDesc,
     kPlayingTimeBottom, k3rdEditColumnLeft},
    kDETNoFlags, kDETNoProperty,
    StaticTextFromView { kDETApplicationFont,
     kDETApplicationFontSize, kDETLeft, kDETNormal, "Minutes" };

    {kPlayingTimeTop, k3rdEditColumnLeft, kPlayingTimeBottom,
     k3rdEditColumnLeft + kEditTextWidth},
    kDETNoFlags, prPlayingTimeSeconds,
    StaticText { kDETApplicationFont, kDETApplicationFontSize,
     kDETLeft, kDETNormal };

    {kPlayingTimeTop, k3rdEditColumnLeft + kSpaceBeforeEditDesc,
     kPlayingTimeBottom, k4thEditColumnLeft},
    kDETNoFlags, kDETNoProperty,
    StaticTextFromView { kDETApplicationFont,
     kDETApplicationFontSize, kDETLeft, kDETNormal, "Seconds" };
    }
};
```

All of the routines you can provide in code resources for aspect templates are described in “Functions You Can Provide as Part of Your Code Resource” beginning on page 5-148. The Catalogs Extension can call the code resource for the aspect of the information page the user is currently using, or it can target another code resource. The code resource in Listing 5-8 handles only calls from the CE that are not targeted or for which the target is

kDETSelf. Targeting of code resource routines is described in "Target Specifier" on page 5-142.

At initialization, Listing 5-8 sets the call-for mask so that the CE calls this code resource only for idle events and view-change events. Thus, the CE calls this code resource periodically to let it process idle-time tasks. The CE also calls this code resource whenever the user opens the information page or displays a conditional view. The call-for mask is described in "Call-For Mask" on page 5-149.

Listing 5-8 calls routines provided by the CE—referred to in this chapter as *callback routines*—to get and set the values of properties and to obtain the number of items in the sublist. The CallBackDET macro that you can use to call these routines is described on "Calling CE-Provided Functions" on page 5-197. All of the available callback routines are described in "CE-Provided Functions That Your Code Resource Can Call" beginning on page 5-196.

Listing 5-8 calculates the playing time by adding up the playing times of all the individual tracks. To calculate this total, Listing 5-8 calls the kDETcmdGetPropertyNumber callback routine repeatedly, targeting each call to the attribute representing a specific track. See the description of the kDETSublistItem target selector in "Target Specifier" on page 5-142 to gain a better understanding of this technique.

IMPORTANT
When you design your code resource, you must follow certain rules to avoid corrupting or crashing the Finder. See "Rules for Writing Code Resources" on page 5-142 for details. ▲

Listing 5-8 Template code resource

```
/* Forward declaration of function defined later */

static OSErr DoIdle(DETCallBlockPtr callBlockPtr);

/* Dispatcher for routines in this code resource that the CE can call */

pascal OSErr MyAlbumCode(DETCallBlockPtr callBlockPtr)
   {
   OSErr err = kDETDidNotHandle;

   if ((callBlockPtr->protoCall.reqFunction < kDETcmdTargetedCall) ||
    (callBlockPtr->protoCall.target.selector == kDETSelf))
      {
      switch (callBlockPtr->protoCall.reqFunction)
         {
         case kDETcmdInit:
            callBlockPtr->init.newCallFors = kDETCallForIdle +
```

```
            kDETCallForViewChanges;
            break;

         case kDETcmdIdle:
         case kDETcmdViewListChanged:
            err = DoIdle(callBlockPtr);
            break;
         }
      }

   return err;
   }

/* This routine calls the CE-provided routine kDETcmdGetPropertyNumber to
    obtain the value of a property as a number. */

static OSErr DoGetPropertyNumber(DETCallBlockPtr callBlockPtr,
                                 DETTargetSelector selector,
                                 long itemNumber,
                                 short property,
                                 long *value)
   {
   OSErr err;
   DETCallBackBlock cbb;

   cbb.getPropertyNumber.reqFunction = kDETcmdGetPropertyNumber;
   cbb.getPropertyNumber.property = property;

   cbb.getPropertyNumber.target.selector = selector;
   cbb.getPropertyNumber.target.aspectName = nil;
   cbb.getPropertyNumber.target.itemNumber = itemNumber;

   err = CallBackDET(callBlockPtr, &cbb);

   *value = cbb.getPropertyNumber.propertyValue;

   return err;
   }

/* This routine calls the CE-provided routine kDETcmdSublistCount to
    obtain the number of items in a sublist. */
```

```
static OSErr DoGetNumSublistItems(DETCallBlockPtr callBlockPtr, long *num)
   {
   OSErr err;
   DETCallBackBlock cbb;

   cbb.sublistCount.reqFunction = kDETcmdSublistCount;
   cbb.sublistCount.target.selector = kDETSelf;

   err = CallBackDET(callBlockPtr, &cbb);

   *num = cbb.sublistCount.count;

   return err;
   }


/* This routine calls the CE-provided routine kDETcmdSetPropertyNumber to
    set the value of a number property. */

static OSErr DoSetPropertyNumber(DETCallBlockPtr callBlockPtr,
                                 short property,
                                 long newValue)
   {
   OSErr err;
   DETCallBackBlock cbb;

   cbb.setPropertyNumber.reqFunction = kDETcmdSetPropertyNumber;
   cbb.setPropertyNumber.property = property;
   cbb.setPropertyNumber.target.selector = kDETSelf;
   cbb.setPropertyNumber.newValue = newValue;

   err = CallBackDET(callBlockPtr, &cbb);

   return err;
   }

/* Here is the main routine for this code resource. This routine counts the
    number of tracks and adds up the total time for the album. */

static OSErr DoIdle(DETCallBlockPtr callBlockPtr)
   {
   OSErr err;
   long oldNumber, actualNumber;
```

```
/* Get the current value of the property prNumTracks. */
err = DoGetPropertyNumber(callBlockPtr, kDETSelf, 0, prNumTracks,
 &oldNumber);

/* Get the number of items in the sublist. Because each sublist item
    represents one track, set the value of prNumTracks equal to the
    number of sublist items. */
if (err == noErr)
   {
   err = DoGetNumSublistItems(callBlockPtr, &actualNumber);
   }

if ((err == noErr) && (oldNumber != actualNumber))
   {
   err = DoSetPropertyNumber(callBlockPtr, prNumTracks, actualNumber);
   }

if (err == noErr)
   {
   long index;
   long oldSeconds, actualSeconds = 0;
   long seconds;
   long minutes;
   long hours;

   /* Calculate the playing time by adding up the playing times of all
       the tracks, calling each track in the sublist in turn. */
   for (index = 1; (err == noErr) && (index <= actualNumber); ++index)
      {
      err = DoGetPropertyNumber(callBlockPtr, kDETSublistItem, index,
       prTrackSeconds, &seconds);

      if (err == noErr)
         {
         err = DoGetPropertyNumber(callBlockPtr, kDETSublistItem, index,
          prTrackMinutes, &minutes);
         }

      if (err == noErr)
         {
         actualSeconds += (minutes * 60 + seconds);
         }
```

```
   }

/* Get the old total playing time. */
if (err == noErr)
   {
   err = DoGetPropertyNumber(callBlockPtr, kDETSelf, 0,
    prPlayingTimeHours, &hours);
   }

if (err == noErr)
   {
   err = DoGetPropertyNumber(callBlockPtr, kDETSelf, 0,
    prPlayingTimeMinutes, &minutes);
   }

if (err == noErr)
   {
   err = DoGetPropertyNumber(callBlockPtr, kDETSelf, 0,
    prPlayingTimeSeconds, &seconds);
   }

/* Now compare the two playing times. If they're different, set
    the properties to equal the new one. */
if (err == noErr)
   {
   oldSeconds = 3600 * hours + 60 * minutes + seconds;

   if (oldSeconds != actualSeconds)
      {
      hours = actualSeconds / 3600;
      err = DoSetPropertyNumber(callBlockPtr, prPlayingTimeHours,
       hours);

      if (err == noErr)
         {
         actualSeconds -= (hours * 3600);
         minutes = actualSeconds / 60;
         err = DoSetPropertyNumber(callBlockPtr, prPlayingTimeMinutes,
          minutes);
         }

      if (err == noErr)
         {
```

```
            actualSeconds -= (minutes * 60);
            err = DoSetPropertyNumber(callBlockPtr, prPlayingTimeSeconds,
             actualSeconds);
            }
         }
      }
   }

return err;
}
```

AOCE Templates Reference

This section describes the file types, template types, and resource types you can use to extend the capabilities of the Catalogs Extension. For complete lists of the resource types required to create each type of AOCE template, see Table 5-1 on page 5-78 for aspect templates, Table 5-10 on page 5-120 for information page templates, Table 5-11 on page 5-138 for forwarder templates, Table 5-12 on page 5-140 for killer templates, and Table 5-13 on page 5-141 for file type templates.

Code resources are described in detail in "Code Resources Reference" beginning on page 5-142.

File and Resource Types Used by the Catalogs Extension

AOCE templates exist as resources in the resource forks of files in the Macintosh file system. A single file can contain multiple templates. The Catalogs Extension looks for files containing templates in the System Extensions folder inside the System Folder. It looks in all files in its list of appropriate file types. Initially, it uses the following file types:

File type	Description
`'detf'`	CE template file. This file type exists solely to hold templates.
`'dsam'`	A CSAM. If you create new templates to support a CSAM, you can make installation easy for users by including the templates in the CSAM file.
`'msam'`	An MSAM. If you create new templates to support an MSAM, you can make installation easy for users by including the templates in the MSAM file.
`'csam'`	A combined CSAM and MSAM. If you create new templates to support a combined CSAM and MSAM, you can make installation easy for users by including the templates in the SAM file.
`'fext'`	Finder extensions. The CE searches only for those Finder extension files that have creator `'adbk'`, which is the CE. This file supplies the templates that come with the CE and that are installed when the CE is installed.

Note
The abbreviation "dsam"—found in the `'dsam'` file type and in function names and data structures in the AOCE interface files—stands for "directory service access module," the name used for catalog service access modules in early versions of the AOCE software. The `'csam'` file type is so named because it implements "combined service access modules." Therefore a file of type `'dsam'` implements a CSAM and a file of type `'csam'` implements both a CSAM and an MSAM.

See the chapter "Service Access Module Setup" in *Inside Macintosh: AOCE Service Access Modules* for descriptions of the templates you must write to support CSAMs and MSAMs. ◆

File type templates can specify additional file types for the CE to search for template resources. You can use this feature to include AOCE templates with extension files of other types.

Within the file, templates consist of sets of associated resources. Each template in the file includes a signature resource, which specifies the type of the template and the base ID of the associated resources. All signature resources include a version number to tell the CE the format of the template. Because these version numbers are automatically included as a part of the Rez template resource definition, you do not need to include them explicitly in your Rez file. Some signature resources also contain additional template-related information.

AOCE templates use the following resource types for template signature resources:

Resource type	Kind of template
`'deta'`	Aspect template
`'deti'`	Information page template
`'detf'`	Forwarder template
`'detk'`	Killer template
`'detx'`	File type template

AOCE templates also use the following types of resources:

Resource type	Description
`'detb'`	Binary data
`'detc'`	Code resource
`'detm'`	Menu entries
`'detn'`	Number (long)
`'detp'`	Reverse-sort properties list
`'dett'`	Lookup table
`'detv'`	View list
`'detw'`	Custom information page window
`'rstr'`	String (`RString`)
`'rst#'`	`RString` array

In addition, icons are composed of a suite of resource types as described in the chapter "Finder Interface" in *Inside Macintosh, Macintosh Toolbox Essentials*.

IMPORTANT
It is very important not to overlap resource IDs from two different templates within the same file. One way to help ensure that they don't is to separate template base IDs by 250. ▲

Template Names

Every template includes a name resource, described in this section.

Each template must have a name so that it can be referred to by other templates. To avoid ambiguity in such cross-references, you should make your template names unique. To ensure uniqueness, you should start the template name with a four-character application signature registered with Macintosh Developer Technical Support.

kDETTemplateName

The template name resource has a resource ID with an offset of `kDETTemplateName` from the template's base (signature) resource ID.

```
resource 'rstr' (rMyBaseID + kDETTemplateName, purgeable) {
   "WAVE This is the name of my template"
};
```

The `'rstr'` resource is defined as follows:

```
type 'rstr' {
   rstring;       /* an RString */
};
```

To ensure the uniqueness of the template name, start it with a four-character application signature registered with Macintosh Developer Technical Support.

Specifying Record and Attribute Types for Templates

Each aspect and information page template applies only to specific types of records or attributes. To specify the types of records and attributes with which it is used, each template must include one or both of the record-type and attribute-type resources.

If your template applies to records, include a `kDETRecordType` resource but no `kDETAttributeType` resource. If your template applies to attributes in records of any type, include a `kDETAttributeType` resource but no `kDETRecordType` resource. Such

a template also supports stand-alone attributes. If your template applies to attributes in a specific type of record, include both a `kDETAttributeType` resource and a `kDETRecordType` resource. The following table summarizes these rules:

`kDETRecordType`	`kDETAttributeType`	Template applies to
Not present	Not present	Nothing (invalid)
`"my rectype"`	Not present	Records of type `"my rectype"`
Not present	`"my attrtype"`	Attributes of type `"my attrtype"` in records of any type or as stand-alone attributes
`"my rectype"`	`"my attrtype"`	Attributes of type `"my attrtype"` only in records of type `"my rectype"`

Note
Specifying both a `kDETAttributeType` resource and a `kDETRecordType` resource does not prevent a user from dragging an attribute from a sublist and dropping it on the desktop or on another object. To control which attribute types can be dragged from a sublist, use a `kDETAspectDragOut` resource (page 5-102). ◆

Note
A stand-alone attribute has a record type formed by concatenating the value of the constant `kAttributeValueRecTypeBody` (aoce Attribute Value), the attribute tag value, and the attribute type of the original attribute (without the attribute type's length or character set). The `RString` structure that holds the record type has a character set that is the same as the character set of the original attribute type. ◆

kDETRecordType

The record-type resource specifies the record type to which an aspect or information page template applies. The record-type resource has a resource ID with an offset of `kDETRecordType` from the template's base resource ID.

```
resource 'rstr' (rMyBaseResourceID + kDETRecordType, purgeable) {
"WAVE record type"
};
```

To ensure the uniqueness of the record type, start it with a four-character application signature registered with Macintosh Developer Technical Support.

kDETAttributeType

The attribute-type resource specifies the attribute type to which an aspect or information page template applies. The attribute-type resource has a resource ID with an offset of `kDETAttributeType` from the template's base resource ID.

```
resource 'rstr' (rMyBaseResourceID + kDETAttributeType,
                  purgeable) {
"WAVE attribute type"
};
```

To ensure the uniqueness of the attribute type, start it with a 4-character application signature registered with Macintosh Developer Technical Support.

kDETAttributeValueTag

The attribute-tag resource specifies the attribute value tag of the attributes to which an aspect or information page template applies. The attribute-tag resource has a resource ID with an offset of `kDETAttributeValueTag` from the template's base resource ID.

```
resource 'detn' (rMyTemplate + kDETAttributeValueTag, purgeable) {
   'bnry'
};
```

The `'detn'` resource type is defined as follows:

```
type 'detn' {
   longInt;
};
```

You can specify this resource only if the template also contains an attribute-type resource. If this resource is present, then the template applies only to attributes that have the specified tag. Because you can define your own attribute value tags, you can use this resource in any way you wish. If this resource is not present, then the template applies to attributes with any tag value.

The attribute-tag resource is useful to improve the efficiency of the display of addresses in a user address information page. Address attributes in user records should have attribute tags set to the value of the address subtype (for example, `'alan'` for LAN-based mail, `'aphn'` for Direct Dialup mail). Address templates should include the attribute-tag resource to specify the tag they work with. This resource allows the Catalogs Extension to determine very quickly the proper template to use with a given address.

Components of Aspect Templates

The primary purpose of an aspect template is to supply information about a record or attribute in an AOCE catalog. Most of this information is in the form of properties used by either an information page field or a sublist in an information page or dNode. Each information page template includes the name of an aspect used to supply properties for its fields and to generate its sublist (if any). In addition, an aspect template may provide information about how to create new items (records or attributes) of the type described by the template, the icon to use in a sublist, and the string to put in the "Kind" column of a sublist.

An aspect template can contain the resources listed in Table 5-1.

Table 5-1 Resources in aspect templates

Resource type	Offset of resource ID from signature resource ID	Purpose of resource
`'deta'`	`0`	Identifies template as aspect and provides a base resource ID. Required for all aspect templates.
`'rstr'`	`kDETTemplateName`	Name of template. Required for all aspect templates.
`'rstr'`	`kDETRecordType`	Type of record to which the template applies. Either this resource, the `kDETAttributeType` resource, or both must be included.
`'rstr'`	`kDETAttributeType`	Type of attribute to which the template applies. Either this resource, the `kDETRecordType` resource, or both must be included.
`'detn'`	`kDETAttributeValueTag`	Attribute tag of attributes to which the template applies. You can provide this resource if you have also provided the `kDETAttributeType` resource. If you don't provide this resource, the template applies to attributes with any tag value.
icon suite	`kDETAspectMainBitmap`	Suite of icons. Neither the code resource nor the user can change these values. A set of icon resources at this offset is required for main aspect templates. (You may also include in any aspect template one or more icon suites with other resource IDs to provide icons for display in the information page.)

Table 5-1 Resources in aspect templates (continued)

Resource type	Offset of resource ID from signature resource ID	Purpose of resource
'rstr'	kDETAspectKind	The kind of record or attribute as shown in a sublist. Neither the code resource nor the user can change this value. You must include this resource in main aspect templates; it is not needed in other aspect templates.
'rst#'	kDETAspectCategory	The names of categories to which the record or attribute belongs. These names are used internally by the CE. They are also displayed to the user if no template includes a corresponding kDETAspectExternalCategory resource. You must include this resource in main aspect templates for records and stand-alone attributes; it is not needed in other aspect templates.
'rst#'	kDETAspectExternalCategory	The names of categories to which the record or attribute belongs. These names are displayed to the user; they must correspond 1:1 to those in the kDETAspectCategory resource. If no template includes this resource, the CE displays the names in the kDETAspectCategory resource to the user. You can include this resource in main aspect templates; it is not needed in other aspect templates.
'detn'	kDETAspectGender	The gender of the kind to display in a sublist for objects of this type. For use with languages that require this information. You can include this resource in main aspect templates when necessary; it is not needed in other aspect templates.
'rstr'	kDETAspectWhatIs	Help-balloon string for objects of the type described by this aspect when they appear in a sublist. You must include this resource in main aspect templates; it is not needed in other aspect templates.
'rstr'	kDETAspectAliasKind	The kind of record or attribute to display in a sublist for aliases to the type of object described by this aspect. You must include this resource in main aspect templates; it is not needed in other aspect templates.

continued

Table 5-1 Resources in aspect templates (continued)

Resource type	Offset of resource ID from signature resource ID	Purpose of resource
'detn'	kDETAspectAliasGender	The gender of the kind to display in a sublist for an alias to an object of this type. For use with languages that require this information. You can include this resource in main aspect templates when necessary; it is not needed in other aspect templates.
'rstr'	kDETAspectAliasWhatIs	Help-balloon string for aliases to objects of the type described by this aspect when the aliases appear in a sublist. You must include this resource in main aspect templates; it is not needed in other aspect templates.
'rstr'	kDETAspectNewMenuName	Text for the New item in the Catalogs menu for records, or for the Add button for the new-attribute-item dialog box. Include this resource only in a main aspect and only if the user is allowed to add a new record or attribute of this type. If you do not include this resource, the user cannot use the Catalogs menu or Add button to add a new object of this type.
'rstr'	kDETAspectNewEntryName	Name given to newly created records of the type described by this aspect. Include this resource only in a main aspect for a record and only if the user is allowed to add a new record of this type. If you do not include this resource, the user cannot use the Catalogs menu to add a new record of this type.
'rstr'	kDETAspectName	Name displayed in the sublist for newly created attributes of the type described by this aspect. Include only in a main aspect for an attribute, only if the user is allowed to add a new attribute of this type, and only if the kDETAspectNewValue resource does not provide a name for the new attribute.
'detb'	kDETAspectNewValue	The concatenation of the 4-byte attribute tag and the attribute value used as an initial value for newly created attributes of the type described by this aspect. Include this resource only in a main aspect for an attribute and only if the user is allowed to add a new attribute of this type. If you do not include this resource, the user cannot use the Add button to add a new attribute of this type.

Table 5-1 Resources in aspect templates (continued)

Resource type	Offset of resource ID from signature resource ID	Purpose of resource
'detn'	kDETAspectSublistOpenOnNew	If you include this resource set to a nonzero number, the CE automatically opens newly created attributes or records of this type. This resource sets the value of the property that has property number kDETAspectSublistOpenOnNew. Your code resource can specify a different resource, overriding the resource in the aspect template (see "Dynamic Creation of Resources" beginning on page 5-154). You can also use the kDETcmdSetPropertyNumber callback routine (page 5-227) to change this property value at any time. You can include this resource only in main aspect templates. This resource is optional.
'detw'	kDETAspectInfoPageCustomWindow	The width, height, and placement of the set of custom information pages that appears if the user opens the catalog object to which this aspect applies. This resource also specifies whether a page-selection pop-up menu should be included in the window. You can include this resource only in main aspect templates. Include only if you do not want to use the default information page window.
'detv'	kDETAspectInfoPageCustomWindow	A view list that describes items to be displayed on every information page that appears if the user opens the catalog object to which this aspect applies. You can include this resource only in main aspect templates. Optional.
'rst#'	kDETAspectRecordDragIn	Types of records the user is allowed to drag and drop on the catalog object to which this aspect applies, paired with the attribute types used to store aliases to these records. When the user drops such a record, the CE creates an alias to the record and stores it in an attribute of the type you specify (unless your code resource takes some other action). Do not include this resource if you do not want to allow the user to drop records on this catalog object.

continued

Table 5-1 Resources in aspect templates (continued)

Resource type	Offset of resource ID from signature resource ID	Purpose of resource
'rst#'	kDETAspectRecordCatDragIn	Categories of records that can be dropped on the catalog object to which this aspect applies, paired with attribute types used to store aliases to these records. When the user drops such a record, the CE creates an alias to the record and stores it in an attribute of the type you specify (unless your code resource takes some other action). Do not include this resource if you do not want to allow the user to drop records on this catalog object.
'rst#'	kDETAspectAttrDragIn	Record and attribute types of attributes the user is allowed to drag and drop on the catalog object to which this aspect applies. When the user drops such an attribute, the CE creates a copy of the attribute (unless your code resource takes some other action). Together with the record type and attribute type of each attribute the user can drop, the resource lists the attribute type the CE should assign to the new copy of the attribute. Do not include this resource if you do not want to allow the user to drop attributes on this catalog object.
'rstr'	kDETAspectDragInString	Prompt string that the CE displays when the user drags and drops an object on the catalog object to which this aspect applies. You do not have to include this resource if your aspect template does not support drops.
'rstr'	kDETAspectDragInVerb	Label for the OK button in the dialog box that the CE might display when the user drags and drops an object on the catalog object to which this aspect applies. You do not have to include this resource if your aspect template does not support drops.
'rstr'	kDETAspectDragInSummary	Short phrase that describes the action of dropping an object on the catalog object to which this aspect applies. The CE can use this phrase in a selection list if more than one aspect can receive the drop.

Table 5-1 Resources in aspect templates (continued)

Resource type	Offset of resource ID from signature resource ID	Purpose of resource
`'rst#'`	`kDETAspectDragOut`	Types of attributes that the user is allowed to drag out of the aspect's sublist. If you do not provide this resource, the user can drag any attribute types from the sublist. To prevent the user from dragging any attributes from the sublist, include this resource but do not specify any attribute types.
`'detn'`, `'rstr'`, `'detb'`	Any property number in the range 0–249	If the aspect template has not used a lookup table or code resource to construct a property with this property number, the CE looks for a resource with an offset equal to this property number and uses the value in that resource as the value of the property. Your code resource can change the value of these resources before the CE loads them (see "Dynamic Creation of Resources" beginning on page 5-154). You can also use callback routines to change property values and types from a code resource at any time (see "Setting Value, Type, and Other Features of Properties" beginning on page 5-223).
`'detm'`	`kDETAspectViewMenu`	A list of property name and property number pairs that specifies the properties by which the user can sort items in a sublist. The property names you list in this resource appear in the View menu, and the property numbers tell the CE what to sort by. If the property number is positive, the sort is alphanumeric; if you specify the negative of the property number, the sort is numeric. You should provide this resource for any aspect template that includes a sublist.
`'detp'`	`kDETAspectReverseSort`	A list of properties that you want sorted in reverse order; that is, descending alphanumeric or ascending numeric order. You can list any property that you have already listed in a `kDETAspectViewMenu` resource.

continued

Table 5-1 Resources in aspect templates (continued)

Resource type	Offset of resource ID from signature resource ID	Purpose of resource
'rst#'	kDETAspectBalloons	Help-balloon strings for views in an information page. You should provide strings for any properties you define, starting with property number kDETFirstDevProperty. For each such property, you include a string to be used if the property is editable and one that appears if the property is not editable. The strings appear in help balloons when Balloon Help is on and the user places the cursor on a view that has a help-balloon property number corresponding to an entry in this resource. If your aspect contains any such properties, you should include this resource.
'dett'	kDETAspectLookup	A lookup table that parses attributes into properties. Lookup tables are described in "The Lookup-Table Resource" beginning on page 5-105. You can include a lookup table in any aspect template.
'detc'	kDETAspectCode	A code resource. Code resources for aspect templates are described in "Code Resources Reference" beginning on page 5-142. You can include a code resource in any aspect template.

Properties

Properties are individual, self-contained pieces of information, such as a number or a string. Properties come from several sources: attribute values, resources in the template, constants, and data provided by the Catalogs Extension. Each property is identified by a property number. The property number uniquely identifies that property within an aspect. Note, however, that other aspects can have entirely unrelated properties with the same number; thus, it is important that each aspect have a unique name.

Each property also has a type. Table 5-2 lists the property types currently defined by Apple Computer, Inc.

Table 5-2 Property types

Property type	Use
`kDETPrTypeNumber`	A number, stored internally as a 32-bit unsigned long word
`kDETPrTypeString`	A string, stored as an `RString` structure
`kDETPrTypeBinary`	A binary block, stored as an uninterpreted sequence of bytes of any size

The CE converts between these types as necessary for purposes of storage or display. For example, a number stored in a 32-bit field in an attribute would be kept in a number-type property when read in to the aspect. However, the number would have to be displayed to the user as a character string. The CE does this conversion automatically.

You can define your own property types as needed. Apple Computer, Inc., reserves all property-type values less than or equal to 0. Therefore, if you define your own property type, give it a positive property-type value. Whenever the CE needs to convert to or from one of your private property types, it calls your code resource. The CE performs such conversions only between string and numeric types.

For example, you might want to store the date and time as a 32-bit long integer. Your code resource could convert between that format and a string. The date and time could then be stored in 32-bit format within the attribute value but displayed to the user using normal edit-text views in an information page.

Some properties are supplied as resources in the aspect template file, some properties are constructed from the contents of attribute values, and some are derived from the catalog system in other ways.

Properties are divided into three main categories:

- Local properties: Properties with property numbers from 0 through 249 are taken from the current (local) aspect. First, the CE checks properties constructed by parsing attribute values. (“The Lookup-Table Resource” beginning on page 5-105 describes how to use lookup tables to convert attribute values to properties.) If the CE finds a constructed property with the appropriate number, it uses that property. If not, then the CE looks for a template resource of type `'detn'` (number), `'rstr'` (type `RString`), or `'detb'` (binary) with a resource ID equal to the template base ID plus the property number. You should use property numbers starting with the value `kDETFirstDevProperty`. For example, property `rMyProperty` of type `RString` might be defined as follows:

```
#define rMyProperty kDETFirstDevProperty + 10

resource 'rstr' (rMyAspectResourceID + rMyProperty, purgeable)
{
"My fixed property value."
};
```

- Constants: information page templates can use property numbers in the range 250–499 as a shortcut to defining numeric constants in the range 0–249. To specify the constant n ($0 \le n \le 249$), use property number $n + 250$.
- Metaproperties: These properties are generated by the CE itself or are retrieved from an AOCE catalog, but they are not attribute values. Examples of these are the name or type of a record, or the value of a record or attribute access mask. Table 5-3 lists the kinds of metaproperties that are available. (Note that a "metaproperty name" is a symbolic name for a reserved property number.)

Several of the metaproperty descriptions in Table 5-3 mention property commands. A ***property command*** is any command handled by your code resource's `kDETcmdPropertyCommand` routine (page 5-159). The CE calls your code resource with the `kDETcmdPropertyCommand` routine selector whenever the user clicks a button or checkbox in your information page, when the user selects an item in a pop-up menu in your information page, and in a few other circumstances, as described in Table 5-14 starting on page 5-161. Each property command includes a property number, usually the number of the property that corresponds to the information page control that originated the command.

Table 5-3 Metaproperties

Metaproperty name	Use
`kDETPrName`	The name of an attribute, record, or alias. If you specify this property in a view list, for example, the CE displays the name of the record, the name of the alias (which it gets from the record pointed to by the alias), or the name of the attribute (which it gets from the `kDETAspectName` property). You cannot store a name in this metaproperty. To store the name of an attribute, use the `kDETAspectName` property in a lookup table or resource.
`kDETPrKind`	The kind of a record or alias.
`kDETPastFirstLookup`	The value of this property is 0 until the CE has completed its first catalog lookup, after which it's 1.
`kDETInfoPageNumber`	The number of the information page currently being displayed. This number is 0 if no information page is open. You can have the CE display a different information page by issuing a property command (such as a command sent by a pop-up menu) with this property number and with the new page number in the `parameter` field. You must use this metaproperty to implement your own page-selection pop-up menu in place of the default pop-up menu.
`kDETAspectTemplateNumber`	The template number of the targeted aspect's template. This value can be used with the `kDETAspectTemplate` target selector (see "Target Specifier" on page 5-142).

Table 5-3 Metaproperties (continued)

Metaproperty name	Use
`kDETInfoPageTemplateNumber`	The template number of the template for the currently open information page (if any). This value can be used with the `kDETInfoPageTemplate` target selector (see "Target Specifier" on page 5-142).
`kDETOpenSelectedItems`	A property number for use with the property command associated with an Open button in an information page with a sublist. When you issue a property command with this property number, the CE opens the sublist items the user has selected.
`kDETAddNewItem`	A property number for use with the property command associated with an Add button in an information page with a sublist. When you issue a property command with this property number, the CE displays a dialog box that lets the user add an attribute to the sublist.
`kDETRemoveSelectedItems`	A property number for use with the property command associated with a Remove button in an information page with a sublist. When you issue a property command with this property number, the CE removes the sublist items the user has selected.
`kDETAspectSublistOpenOnNew`	If you set this property to a nonzero number, the CE automatically opens newly created sublist entries. You can also set a default value for this property by including a `'detn'` resource at this offset in the aspect template.
`kDETDNodeAccessMask`	The dNode access mask.
`kDETRecordAccessMask`	The record access mask.
`kDETAttributeAccessMask`	The attribute access mask.
`kDETPrimaryMaskByBit`	The full set of 16 bits of the primary access mask (the access mask for the object you are in); the following properties provide values for several of the individual bits.
`kDETPrimarySeeMask`	The "see" access mask bit (1 for "can see")
`kDETPrimaryAddMask`	The "add" access mask bit (1 for "can add")
`kDETPrimaryDeleteMask`	The "delete" access mask bit (1 for "can delete")
`kDETPrimaryChangeMask`	The "change" access mask bit (1 for "can change")
`kDETPrimaryRenameMask`	The "rename" access mask bit (1 for "can rename")
`kDETPrimaryChangePrivsMask`	The "change privileges" access mask bit (1 for "can change privileges")

NOTE Access masks are described in the chapter "Catalog Manager" in this book.

Aspect Template Signature Resource

As with all templates, an aspect template has a signature resource that identifies the type of template and that provides a base resource ID relative to which all other resource IDs are numbered.

'deta' Resource

The signature resource for an aspect template is of type `'deta'`.

```
type 'deta' {
   longInt = kDETAspectVersion;                 /* template format version */
   longInt;                                     /* drop-operation order */
   boolean dropCheckConflicts, dropCheckAlways; /* confirm drops? */
   boolean notMainAspect, isMainAspect;         /* main aspect? */
   align word;                                  /* reserved */
};
```

The Catalogs Extension uses the drop-operation order in case a catalog object is associated with more than one aspect that can handle drop operations. The operation specified by the aspect with the lowest drop-operation order is offered to the user. If two templates have the same drop-operation order number, the CE picks one arbitrarily. If your template does not support drop operations, the CE ignores this field.

If you specify `dropCheckAlways` as the drop-check Boolean value, the CE always displays a dialog box asking the user to confirm a drop operation before performing it. If you specify `dropCheckConflicts` as this Boolean value, the CE displays the dialog box only if there is more than one aspect that supports drops for this catalog object. You must provide a prompt string and button label for the dialog box in additional resources in the aspect template. If your template does not support drop operations, the CE ignores this field.

See “Drags and Drops” on page 5-28 for more information about drag-and-drop operations. See “Supporting Drags and Drops” beginning on page 5-98 for descriptions of all the resources you must provide to support these operations.

You use the second Boolean field of the aspect signature resource to specify whether this template is for a main aspect. The next section describes additional resources you must provide if this is a main aspect template.

Main Aspect Template Resources

Main aspects provide the information the Catalogs Extension needs to display the record or attribute value in a sublist—such as a record in a dNode window or an attribute value in a record window. Figure 5-21 on page 5-44 shows an information page with a sublist. Main aspect templates provide the information the CE needs to create a main aspect. You

can also use a main aspect template to specify the size, location, and standard contents of custom information pages that the CE displays when the user opens an object to which the main aspect template applies.

Table 5-4 lists the resources used specifically by main aspect templates. In addition to the resources in Table 5-4, main aspect templates can include any of the resources shown in Table 5-1 on page 5-78. The resources required by main aspect templates are described in this section. The resources required for all aspect templates are described in the preceding sections. All aspect-template resource descriptions are summarized in Table 5-1 on page 5-78.

Note that main aspects can support information pages in addition to supporting a sublist. For example, a single main aspect can provide the information needed to list a specific attribute value in a sublist, plus all the properties needed by the information page displayed when the user double-clicks that line on the sublist.

Table 5-4 Resources used by main aspect templates

Resource type	Offset of resource ID from signature resource ID	Comments
`'deta'`	`0`	Required for all aspect templates.
`'rstr'`	`kDETTemplateName`	Required for all aspect templates.
`'rstr'`	`kDETRecordType`	Either this resource, the `kDETAttributeType` resource, or both must be included.
`'rstr'`	`kDETAttributeType`	Either this resource, the `kDETRecordType` resource, or both must be included.
icon suite	`kDETAspectMainBitmap`	Required for all main aspect templates.
`'rstr'`	`kDETAspectKind`	Required for all main aspect templates.
`'rst#'`	`kDETAspectCategory`	Required for records and stand-alone attributes.
`'rst#'`	`kDETAspectExternalCategory`	Optional.
`'detn'`	`kDETAspectGender`	Optional.
`'rstr'`	`kDETAspectWhatIs`	Required for all main aspect templates.
`'rstr'`	`kDETAspectAliasKind`	Required for all main aspect templates.
`'detn'`	`kDETAspectAliasGender`	Optional.
`'rstr'`	`kDETAspectAliasWhatIs`	Required for all main aspect templates.
`'rstr'`	`kDETAspectNewMenuName`	Include if the user is allowed to add a new record or attribute of this type.

continued

Table 5-4 Resources used by main aspect templates (continued)

Resource type	Offset of resource ID from signature resource ID	Comments
'rstr'	kDETAspectNewEntryName	Include in a template for a record if the user is allowed to add a new record of this type.
'rstr'	kDETAspectName	Include in a template for an attribute if the user is allowed to add a new attribute of this type, and if the kDETAspectNewValue resource does not provide a name for the new attribute.
'detb'	kDETAspectNewValue	Include in a template for an attribute if the user is allowed to add a new attribute of this type.
'detn'	kDETAspectSublistOpenOnNew	Optional.
'detw'	kDETAspectInfoPageCustomWindow	Include only if you do not want to use the default information page window.
'detv'	kDETAspectInfoPageCustomWindow	Optional.

kDETAspectMainBitmap

A main aspect template must include an icon suite with a resource ID that has an offset of kDETAspectMainBitmap from the template's base resource ID. Suppose, for example, that you prepared an icon suite in a ResEdit file named AlbumIcons, that all of your icon resources had resource IDs of 0, and that your resource base ID was kMainAspect. In that case, you could use the following code to include the icon suite in your main aspect template:

```
include "AlbumIcons" 'ICN#'(0) as
                  'ICN#'(kMainAspect + kDETAspectMainBitmap, purgeable);
include "AlbumIcons" 'icl4'(0) as
                  'icl4'(kMainAspect + kDETAspectMainBitmap, purgeable);
include "AlbumIcons" 'icl8'(0) as
                  'icl8'(kMainAspect + kDETAspectMainBitmap, purgeable);
include "AlbumIcons" 'ics#'(0) as
                  'ics#'(kMainAspect + kDETAspectMainBitmap, purgeable);
include "AlbumIcons" 'ics4'(0) as
                  'ics4'(kMainAspect + kDETAspectMainBitmap, purgeable);
include "AlbumIcons" 'ics8'(0) as
                  'ics8'(kMainAspect + kDETAspectMainBitmap, purgeable);
include "AlbumIcons" 'SICN'(0) as
                  'SICN'(kMainAspect + kDETAspectMainBitmap, purgeable);
```

The icon suite must be included in the main aspect template or specified by your `kDETcmdDynamicResource` code resource routine (page 5-156). Once the icon suite has been specified, it cannot be changed from a code resource or by the user.

kDETAspectKind

Specify the kind of the record or attribute as it is to be displayed in a sublist with an `RString` resource with an offset of `kDETAspectKind` from the template's base resource ID.

```
resource 'rstr' (kMainAspect + kDETAspectKind, purgeable)
{
   "My Kind"
};
```

This resource must be included in the main aspect template or specified by your `kDETcmdDynamicResource` code resource routine (page 5-156). Once the record or attribute kind has been specified, it cannot be changed from a code resource or by the user.

kDETAspectCategory

Specify the categories to which the record or attribute belongs with an `RString`-array resource with an offset of `kDETAspectCategory` from the template's base resource ID.

```
resource 'rst#' (kMainAspect + kDETAspectCategory, purgeable)
{
   { "Category 1", "Category 2" }
};
```

The `'rst#'` resource type is defined as follows:

```
type 'rst#' {
   integer = (endOfData - startOfData) / 8;
startOfData:
   integer = $$CountOf(RStrArray);          /* Array size */
   array RStrArray {
      rstring;
      align word;
   };
endOfData:
};
```

Every record must be assigned to one or more categories. The Catalogs Extension uses record categories to group records in catalog windows. You also specify record categories if you include a `kDETAspectRecordCatDragIn` resource in your template (see page 5-99). If no template includes a `kDETAspectExternalCategory` resource, the CE uses the category name you provide in the `kDETAspectCategory` resource for display in the View menu.

You must include a `kDETAspectCategory` resource in an attribute's main aspect template if the template supports a stand-alone attribute—that is, if you do not include a `kDETRecordType` resource in the main aspect template.

Note that an item in a sublist can be of only one kind, but it can be in as many categories as is appropriate. Thus, the kind resource is a single string of type `RString`, but the category resource is an array of `RString` strings.

kDETAspectExternalCategory

Specify category names to be displayed to the user with an `RString`-array resource with an offset of `kDETAspectExternalCategory` from the template's base resource ID.

```
resource 'rst#' (kMainAspect + kDETAspectExternalCategory, purgeable)
{
   { "Category 1", "Category 2" }
};
```

This resource must contain one category name for each name in the `kDETAspectCategory` resource, in the same sequence. If you do not include this resource in the main aspect template, the Catalogs Extension uses the external category names provided for these categories in any other template available. If no template provides this resource, the CE uses the names in the `kDETAspectCategory` resource. (If more than one template provides a `kDETAspectExternalCategory` resource, the CE picks one of them and ignores the others.)

You should use category names in your local language for the `kDETAspectCategory` resource when you are creating the template. Then, when someone localizes the template to another language, the person doing the localizing can add a `kDETAspectExternalCategory` template to change the names displayed to the user.

kDETAspectGender

You can specify the gender of the record kind of the record to which this main aspect applies with a resource of type `'detn'` with an offset of `kDETAspectGender` from the template's base resource ID.

```
resource 'detn' (kMainAspect + kDETAspectGender, purgeable) {
   1
};
```

You can use this resource to match the article of the record kind to the gender of the record kind when you are displaying the kind to the user. The significance of the value in this resource depends on the language with which it is being used; you can place any value you wish in this resource, and you are responsible for interpreting it. It is recommended that you follow the guidelines in *Guide to Macintosh Software Localization.*

kDETAspectWhatIs

Each main aspect template must provide a help-balloon string. The Finder displays this string when the user enables Balloon Help online assistance and moves the cursor over an object (a record or attribute) of the type to which this main aspect applies. The help-balloon string is in an `RString` resource with an offset `kDETAspectWhatIs` from the template's base resource ID.

```
resource 'rstr' (kMainAspect + kDETAspectWhatIs, purgeable) {
   "Flue handle record\n\nA record containing information about a flue
handle."
};
```

kDETAspectAliasKind

To specify the kind of an alias to a record or attribute as it is to be displayed in a sublist, use an `RString` resource with an offset of `kDETAspectAliasKind` from the template's base resource ID.

```
resource 'rstr' (kMainAspect + kDETAspectAliasKind, purgeable)
{
   "My kind alias"
};
```

kDETAspectAliasGender

To specify the gender of the kind of the alias to a record, use a resource of type `'detn'` with an offset of `kDETAspectAliasGender` from the template's base resource ID.

```
resource 'detn' (kMainAspect + kDETAspectAliasGender, purgeable) {
   1
};
```

The significance of the value in this resource depends on the language with which it is being used; you can place any value you wish in this resource, and you are responsible for interpreting it.

kDETAspectAliasWhatIs

Each main aspect template must provide a help-balloon string for aliases. The Finder displays this string when the user enables Balloon Help online assistance and moves the cursor over an alias to an object (a record or attribute) of the type to which this main aspect applies. The help-balloon string is in an `RString` resource with an offset `kDETAspectAliasWhatIs` from the template's base resource ID.

```
resource 'rstr' (kMainAspect + kDETAspectAliasWhatIs, purgeable) {
   "Alias to a flue handle record\n\nAn alias to a record containing
information about a flue handle."
};
```

kDETAspectNewMenuName

A main aspect template for a record type specifies how new records of that type are to be added to the containing dNode. Similarly, a main aspect template for an attribute type specifies how new attributes of that type are to be added to the containing record. To allow the user to add a new record or attribute, you must supply an `RString` resource containing the text of the New item for the Catalogs menu or the Add item for the new-attribute dialog box. The menu name resource must have an ID with the offset `kDETAspectNewMenuName` from the template's base resource ID.

```
resource 'rstr' (rMyAspectResourceID + kDETAspectNewMenuName, purgeable)
{
   "The text for the New menu item goes here"
};
```

IMPORTANT
To allow the user to add a new attribute, you must provide a New button in your information page, and the property command associated with that button must use the property number `kDETAddNewItem`. See Table 5-3 on page 5-86 for a description of this and other special property numbers. ▲

kDETAspectNewEntryName

A main aspect template for a record type must also specify the name the Catalogs Extension should give to newly created records of that type. To provide this name, use an `RString` resource with an offset of `kDETAspectNewEntryName` from the template's base resource ID.

```
resource 'rstr' (rMyAspectResourceID + kDETAspectNewEntryName, purgeable)
{
   "Name of new record goes here"
};
```

kDETAspectName

A main aspect template for an attribute type can specify a default name for newly created attribute values of that type. To provide this name, use an `RString` resource with an offset of `kDETAspectName` from the template's base resource ID.

```
resource 'rstr' (rMyAspectResourceID + kDETAspectName, purgeable)
{
   "Name of new attribute goes here"
};
```

For main aspect templates for records, the Catalogs Extension automatically sets the `kDETAspectName` property to be the name of the record. This property provides the name the CE displays in the "Name" column in dNode windows.

If you wish to specify a name for an attribute value, you can provide a value for the `kDETAspectName` property in the main aspect template for the attribute. You can display attribute value names in sublists (use the metaproperty `kDETPrName` for this purpose); the CE uses attribute value names for stand-alone attributes.

You can store the name of an attribute in the `kDETAspectName` property at any time. You can use the lookup table, a code resource, a `kDETAspectName` resource, or any combination of these methods to provide a value for the `kDETAspectName` property.

You should limit sublist items to one line. Multiline sublist items are not guaranteed to work correctly.

Note
For records to be displayed in the Key Chain, your setup main aspect template must provide a `kDETAspectName` resource to explicitly set this property. See the chapter "Service Access Module Setup" in *Inside Macintosh: Service Access Modules* for complete information about setup templates. ◆

kDETAspectNewValue

If the new item is an attribute value, the main aspect template must contain a binary block resource (type `'detb'`) containing the concatenation of the attribute tag and the new value. Give this resource an ID with an offset of `kDETAspectNewValue` from the template's base resource ID.

```
data 'detb' (kTrackAspect + kDETAspectNewValue, purgeable) {
   $"626E 7279"                          // tag (bnry)
   $"0000 0001"                          // prTrackNumber (1)
   $"0000 0000"                          // prTrackMinutes (1)
   $"0000 0000"                          // prTrackSeconds (1)
   $"0000 0007 3C74 6974 6C65 3E"        // kDETAspectName (<title>)
   $"0000 000A 3C63 6F6D 706F 7365 723E" // composer (<composer>)
   $"0000 000A 3C63 6F6D 6D65 6E74 733E" // comments (<comments>)
```

The Catalogs Extension needs the attribute tag, the attribute value, and the attribute type to add an attribute to a record in an AOCE catalog. When the user clicks the Add button in your information page to create a new attribute value, the CE gets the attribute type from the aspect template's attribute-type resource (see "Specifying Record and Attribute Types for Templates" on page 5-75) and the tag and initial attribute value from the `kDETAspectNewValue` resource. Once the CE has added the attribute to the catalog record, the CE uses the main aspect for attributes of that type to display that attribute in a sublist.

kDETAspectSublistOpenOnNew

A main aspect template can specify whether the Catalogs Extension should automatically open an information page for a newly created attribute or record in a sublist. If you set the value of property `kDETAspectSublistOpenOnNew` to a nonzero number, the CE automatically opens newly created attributes or records of the type associated with the main aspect. You can provide a `'detn'` resource with an offset of `kDETAspectSublistOpenOnNew` to provide a default value for this property.

```
resource 'detn' (kMainAspect + kDETAspectSublistOpenOnNew, purgeable) {
   1
};
```

Your code resource can change the value of this resource before the CE loads it (see "Dynamic Creation of Resources" beginning on page 5-154). You can also use the `kDETcmdSetPropertyNumber` callback routine (page 5-227) to change this property value from a code resource at any time. This resource is optional.

kDETAspectInfoPageCustomWindow

If you want to specify a nonstandard size or location for the information page window that appears when the user opens the catalog object to which this main aspect applies, you can provide a `'detw'` resource with an offset of `kDETAspectInfoPageCustomWindow` from the template's base resource ID. This resource also specifies whether a page-selection pop-up menu should be included in the window.

```
resource 'detw' (kMainAspect + kDETAspectInfoPageCustomWindow, purgeable) {
   {kCustomPageTop, kCustomPageLeft, kCustomPageBottom, kCustomPageRight},
   discludePopup
};
```

The `'detw'` resource type is defined as follows:

```
type 'detw' {
   rect;                                /* info-page window in  */
                                        /*  global coordinates */
   boolean discludePopup, includePopup;/* include a page-selection pop-up? */
   align word;                          /* Future expansion */
};
```

Note that you can specify no pop-up menu only if there is only one information page for the object or if you are also providing a `'detv'` resource in this main aspect template that provides a pop-up menu of information page names.

If there is more than one information page for the object and you include this resource in the main aspect template, then all of the information pages have the size you specify in this resource.

IMPORTANT

If the information page window you specify is too large to be displayed on the user's screen, the Finder makes the window smaller, truncating the bottom and right side of the information page. To be sure an information page can fit on a Macintosh computer screen of any size, make it no larger than 322 pixels high by 512 pixels wide. ▲

There are two special values you can use for the top left corner of a custom information page window:

- specify (0, 0) to place the upper-left corner of the window slightly below and to the right of the upper-left corner of the parent window
- specify (–1, –1) to center the window on the screen

If you want to specify a view list for views that are to be displayed on all the information pages that appear when the user opens the catalog object to which this main aspect applies, you can provide a `'detv'` resource with an offset of `kDETAspectInfoPageCustomWindow` from the template's base resource ID.

If, as part of this view list, you include a pop-up menu (view type `Menu`) with a property number of `kDETInfoPageNumber`, the Catalogs Extension displays a pop-up menu with a list of information page names at the location you specify. Therefore, by combining a `'detw'` resource with the `discludePopup` flag set and a `'detv'` resource with a pop-up menu for selecting information pages, you can create a custom set of information pages with the pop-up menu at any location you wish. The following view list displays an icon and a pop-up menu for selecting information pages.

```
resource 'detv' (kMainAspect + kDETAspectInfoPageCustomWindow, purgeable) {
   {
   {kBitmapTop, kBitmapLeft, kBitmapBottom, kBitmapRight},
   kDETNoFlags, kDETAspectMainBitmap,
   Bitmap { kDETLargeIcon };

   {kMenuTop, kMenuLeft, kMenuBottom, kMenuRight},
   kDETPopupDynamicSize, kDETInfoPageNumber,
   Menu { kDETApplicationFont, kDETApplicationFontSize,
                  kDETLeft, kDETNormal, "Page", kDETInfoPageNumber,
                  rMenuResourceID };
   }
};
```

View lists are described in "View Lists" beginning on page 5-123.

Supporting Drags and Drops

If your aspect supports drags and drops, your template should include the resources listed in this section, a code resource that handles drags and drops, or both. Drags and drops are described in "Drags and Drops" on page 5-28. For more information on how code resources handle drags and drops, see the descriptions of the `kDETcmdDropQuery` (page 5-172) and `kDETcmdDropMeQuery` (page 5-170) routines.

kDETAspectRecordDragIn

When a user drags a record and drops it on another record, the most common result is that the Catalogs Extension creates an alias to the record and stores it in an attribute in the target record. When the user drops a record on a record for which you provided an aspect template, the CE looks for a resource with an ID offset of `kDETAspectRecordDragIn` from the base resource ID. This resource lists the types of records that can be dragged into the aspect, paired with the attribute type in which the CE should store an alias to the record that was dragged. Use the `kDETAspectRecordCatDragIn` resource (described next) to specify categories of records that can be dragged and dropped on the catalog object.

```
resource 'rst#' (kMyAspectTemplate + kDETAspectRecordDragIn, purgeable)
{
    "Record type 1", "Alias attribute type 1",
    "Record type 2", "Alias attribute type 2"
};
```

Do not include a `kDETAspectRecordDragIn` resource if you do not want to allow the user to drop records on this catalog object.

Note

In addition to checking for drag-in resources, the CE calls your code resource (if any) to check whether the code resource can handle the drop. A code resource can take an action different from that specified in the drag-in resources, can handle a drop in the absence of drag-in resources, and can handle drags and drops for source and destination objects other than records (as long as either the source or the destination is an AOCE catalog object). ◆

See the discussion of drags and drops in "Drags and Drops" on page 5-28 and the descriptions of the `kDETcmdDropQuery` (page 5-172) and `kDETcmdDropMeQuery` (page 5-170) routines for more information on this process.

IMPORTANT

If your code resource does not take a different action and you do want the CE to store an alias to the record in an attribute value, you must be sure that you have listed the attribute type in your aspect's lookup table with the `useInSublist` and `isAlias` flags set. Lookup tables are described in "The Lookup-Table Resource" beginning on page 5-105. ▲

kDETAspectRecordCatDragIn

Instead of or in addition to specifying individual record types of records that can be dragged and dropped on the record to which the aspect applies, you can specify categories of records that can be dropped. When the user drops such a record, the

Catalogs Extension creates an alias to the record and stores it in an attribute (unless your code resource takes some other action). For each category of record the user can drop, the resource also lists the attribute type of the new attribute containing the alias to the record. Do not include this resource if you do not want to allow the user to drop records on this catalog object.

```
resource 'rst#' (kMyAspectTemplate + kDETAspectRecordCatDragIn, purgeable)
{
    "Category 1", "Alias attribute type 1",
    "Category 2", "Alias attribute type 2"
};
```

See the preceding discussion of the `kDETAspectRecordDragIn` resource for more information on drags and drops.

kDETAspectAttrDragIn

When a user drags an attribute and drops it on a record, the most common result is that the Catalogs Extension creates a copy of the attribute and stores it in the target record. When the user drops an attribute on a record for which you provided an aspect template, the CE looks for a resource with an ID offset of `kDETAspectAttrDragIn` from the base resource ID. This resource lists the types of records that can be dragged from, the types of attributes that can be dragged into the target record, and the attribute type the CE should assign to the new copy of the attribute. To indicate any type, you can use the empty string "" for the type of record that can be dragged from.

```
resource 'rst#' (kMyAspectTemplate + kDETAspectAttrDragIn, purgeable)
{
    "Record type 1", "Attribute type 1", "New attribute type 1",
    "Record type 2", "Attribute type 2", "New attribute type 2"
};
```

Do not include this resource if you do not want to allow the user to drop attribute values on this catalog object.

Note
In addition to checking for drag-in resources, the CE calls your code resource (if any) to check whether the code resource can handle the drop. A code resource can take an action different from that specified in the drag-in resources, can handle a drop in the absence of drag-in resources, and can handle drags and drops for source objects other than attributes or records and destination objects other than records (as long as either the source or the destination is an AOCE catalog object). ◆

See the discussion of drags and drops in "Drags and Drops" on page 5-28 and the descriptions of the `kDETcmdDropQuery` (page 5-172) and `kDETcmdDropMeQuery` (page 5-170) routines for more information on this process.

kDETAspectDragInString

If your aspect supports drops, you must provide a prompt string for the Catalogs Extension to display when the user drags and drops an object on the catalog object to which the aspect applies. You provide this string in an `RString` resource with an ID offset of `kDETAspectDragInString` from the base resource ID.

```
resource 'rstr' (rMyAspectTemplate + kDETAspectDragInString, purgeable)
{
   "Do you want to send %3%“^3”%the selected items% to *0x/the/* ^1 “^2”?"
};
```

The string `%3%“^3”%the selected items%` helps the CE either insert the name of the item being dragged (in `^3` if it's a single selection) or insert the substring "`the selected items`" (if it's a multiple-item selection).

The token `^1` is the destination's kind as displayed in the sublist (`kDETAspectKind`).

The token `^2` is the destination's name.

When localizing this resource, replace the string `*0x/the/*` with an article consistent with the destination's kind (token `^1`).

You do not have to include this resource if your aspect template does not support drops.

kDETAspectDragInVerb

If your aspect supports drops, you must provide a label for the OK button in the dialog box that the Catalogs Extension can display when the user drags and drops an object on the catalog object to which the aspect applies. You provide this label in an `RString` resource with an ID offset of `kDETAspectDragInVerb` from the base resource ID.

```
resource 'rstr' (rMyAspectTemplate + kDETAspectDragInVerb, purgeable)
{
   "Send"
};
```

If you are unsure as to what label to use for this feature, use OK.

You do not have to include this resource if your aspect template does not support drops.

Drag-in verbs are not implemented in the initial release of the AOCE software, but you should include this resource to support future enhancements to the software.

kDETAspectDragInSummary

If your aspect supports drops, you should provide a short phrase that describes the action of dropping an object on the catalog object to which this aspect applies. The Catalogs Extension can use this phrase in a selection list if more than one aspect can receive the drop.

```
resource 'rstr' (rMyAspectTemplate + kDETAspectDragInSummary, purgeable)
{
   "Send item"
};
```

You do not have to include this resource if your aspect template does not support drops.

Selection lists are not implemented in the initial release of the AOCE software, but you should include this resource to support future enhancements to the software.

kDETAspectDragOut

If your aspect template's lookup table includes any attribute types for which you have set the `useInSublist` flag, the user might try to drag an attribute of that type from the sublist to drop it on another object or on the desktop. You can include a resource of type `'rst#'` that specifies which attribute types the user is allowed to drag from the sublist. You must give this resource an ID offset of `kDETAspectDragOut` from the base resource ID.

```
resource 'rst#' (rMyAspectTemplate + kDETAspectDragOut, purgeable)
{
   {
   "First attribute type",
   "Second attribute type"
   }
};
```

If you do not provide this resource, the user can drag any attribute types from the sublist. To prevent the user from dragging any attributes from the sublist, include this resource but do not specify any attribute types.

Other Aspect Template Resources

Any aspect template can include the resources listed in this section. In addition to the resources described here, see Table 5-1 on page 5-78 for a summary of all of the resources that you can include in an aspect template.

Any property number in the range 0–249

If the information page template has not used a lookup table or code resource to construct a property for any property number in the range 0–249, the Catalogs Extension looks for a resource whose ID has an offset from the base resource ID equal to that property number and uses the value in that resource as the value of the property. You can use resources of types `'detn'`, `'rstr'`, or `'detb'` for this purpose.

You must be careful to ensure that none of your property numbers conflict with the resource ID numbers defined in the AOCE header files. To do so, use the value `kDETFirstDevProperty` for your first property number and increment each additional property number by 1.

kDETAspectViewMenu

The user can sort a sublist in an information page by any of a number of different properties, just as users can sort lists in the Finder by name, kind, date, and so forth. The user can use the View menu to select the property to use for sorting. If your aspect template supports a sublist (that is, if the lookup table includes any attribute types for which you have set the `useInSublist` flag), you should provide a resource of type `'detm'` to specify the properties that can be used for sorting. You must give this resource an ID offset of `kDETAspectViewMenu` from the base resource ID.

```
resource 'detm' (rMyAspectTemplate + kDETAspectViewMenu, purgeable)
{
   rMyAspectResourceID + kDETAspectViewMenu,
   {
   kPrName, "By Name";
   -kPrAge, "By Age";
   }
};
```

The 'detm' resource type is defined as follows:

```
type 'detm' as 'fmnu';
```

For each property by which the sublist may be sorted, you must provide the property number followed by the text that appears in the View menu.

Normally, the Catalogs Extension sorts items in ascending alphanumeric order. To have the items sorted numerically rather than alphanumerically (that is, taking the value of the property as a number rather than a text string), use the negative of the property number. Numeric sorting is descending by default (matching the Finder's normal procedure of sorting sizes and dates in descending order).

For a way to let users sort items in a sublist without using the View menu, see the description of the StaticCommandTextFromView view type on page 5-128.

kDETAspectReverseSort

If you want to sort any properties of items in a sublist in reverse order—that is, descending alphanumeric or ascending numeric order— you must list its property number in a resource of type 'detp'. You must give this resource an ID offset of kDETAspectReverseSort from the base resource ID. The resource can list any property that you have already listed in a kDETAspectViewMenu resource.

```
resource 'detp' (rMyAspectTemplate + kDETAspectReverseSort, purgeable)
{
   { kPrAge }
};
```

The 'detp' resource type is defined as follows:

```
type 'detp' {
   integer = $$CountOf(SortArray);  /* Number of items */
   array SortArray {
      integer;                      /* Property number */
   };
};
```

kDETAspectBalloons

Your aspect template should provide help-balloon strings for any properties that can appear in an information page. The Finder displays a help-balloon string when the user turns on Balloon Help assistance and positions the mouse over a view. For each property you define, you should provide two strings: one to be presented if the property is editable and one to be presented if the property is not editable. The first pair of strings in the resource is used for property number `kDETFirstDevProperty`; the second pair of strings is for property number `kDETFirstDevProperty` + 1; and so forth.

Use a resource of type `'rst#'` to specify the help-balloon strings. You must give this resource an ID offset of `kDETAspectBalloons` from the base resource ID.

```
resource 'rst#' (rMyAspectResourceID + kDETAspectBalloons, purgeable)
{
   {
   "The foobar's age.",
   "The foobar's age. Uneditable because the foob is locked or access is
    restricted.",
   "The foobar's size.",
   "The foobar's size. Uneditable because the foob is locked or access is
    restricted."
   }
}
```

The Lookup-Table Resource

You can use a lookup-table resource in an aspect template to parse attribute values into properties and properties into attribute values. An aspect-template lookup table contains an entry for each type of attribute value to be translated into and from properties. Attribute values to be translated into properties come from two sources:

- Attribute values in the record or attribute to which the aspect applies. For each attribute type for which the lookup table contains a pattern, the lookup table automatically processes all of the attribute values with that attribute type in this record or attribute.
- Attribute values sent to the lookup table by the code resource. You can use the code resource callback routine `kDETcmdBreakAttribute` (page 5-224) to send to the lookup table an attribute value from anywhere within or outside of an AOCE catalog.

Each lookup-table entry includes a list of attribute types, an attribute tag, a flags field, and a pattern that specifies the mapping between attribute values and properties. The attribute types and tag specify the types of attribute values to be processed. Use a tag value of 0 to process all tag types. The flags further qualify how and when the table entry should be used. The lookup-table flags are shown in Table 5-5 on page 5-109.

In many cases, the translation pattern consists of a single item—indicating that the attribute value maps to a single property. However, it is possible to have much more complex patterns, including variable-length and repeating elements. Figure 5-24 on page 5-107 illustrates the format of a lookup table. There is one entry for each type of data block to be parsed (that is, each attribute value with a specific combination of attribute type and attribute value tag), and each entry contains a list of pattern elements. Each pattern element contains three parts: a format, which drives the parsing process; a property number, telling where to store the result (which may be `kDETNoProperty` if no result should be stored); and an "extra" parameter, which is used by some of the pattern types to specify a second property number. Basic lookup-table pattern elements are shown in Table 5-6 on page 5-111.

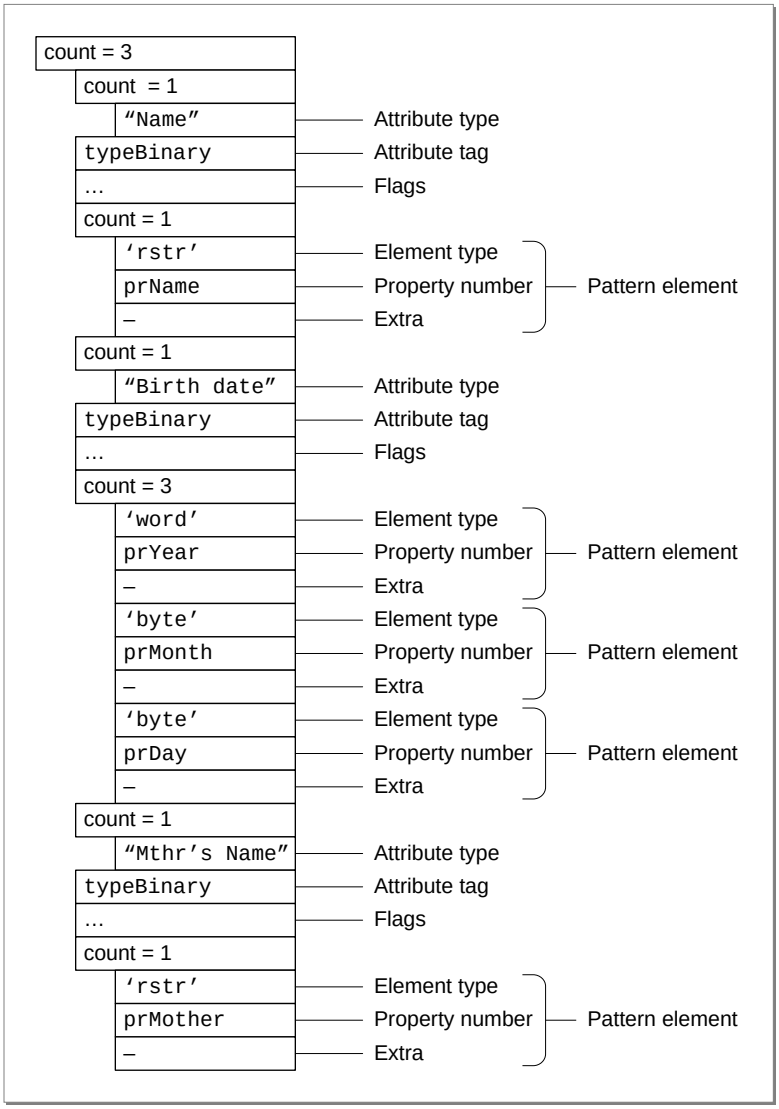
Note that a specific attribute might contain more than one attribute value with the same tag, and you might want to parse these values differently. The lookup-table format includes conditional elements and pattern elements that let you identify each kind of attribute value and process each one appropriately. Lookup-table elements for repeating patterns are shown in Table 5-9 on page 5-115, and elements for conditional patterns are shown in Table 5-7 on page 5-112.

The Catalogs Extension sends any pattern element whose type begins with an uppercase letter to the code resource for processing; see the description of the `kDETcmdPatternIn` routine on page 5-182 for details.

When the CE uses lookup-table patterns to create or update attribute values, a pattern entry can combine any number of property values into a single attribute value. When the user closes an information page, the CE checks whether any properties have changed. If it finds one that has, the CE calls the aspect's code resource (if there is one) with the `kDETcmdValidateSave` routine selector (page 5-168). If the code resource does not want the new property value saved, it returns an error. If the code resource returns `noErr` or `kDETDidNotHandle`, the CE processes all the entries in the lookup table that have the `useForOutput` flag set and that include the property that has changed.

You can provide separate lookup-table entries for input (that is, converting attribute values into property values) and output (converting property values into attribute values), or you can specify that a single entry be used for both purposes.

Figure 5-24 Lookup-table format



kDETAspectLookup

You specify a lookup table with a resource of type `'dett'` with an offset of `kDETAspectLookup` from the template's base resource ID.

```
resource 'dett' (rMyAspectResourceID + kDETAspectLookup, purgeable) {
   {
   {"Name"}, typeBinary,
   useForInput, useForOutput, notInSublist, isNotAlias, isNotRecordRef,
      {
      'rstr', prName, 0;         // name
      };
   {"Birthdate"}, typeBinary,
   useForInput, useForOutput, notInSublist, isNotAlias, isNotRecordRef,
      {
      'word', prYear, 0;         // year of birth
      'byte', prMonth, 0;        // month of birth
      'byte', prDay, 0;          // day of birth
      };
   {"Mthr's Name"}, typeBinary,
   useForInput, useForOutput, notInSublist, isNotAlias, isNotRecordRef,
      {
      'rstr', prMother, 0;       // mother's name
      };
   }
};
```

The format for a lookup-table resource is as follows:

```
type 'dett' {
   integer = $$CountOf(AttributeArray);   /* attribute array size */
   array AttributeArray {
      integer = $$CountOf(TypeArray);     /* attribute type array size */
      array TypeArray {
         RString[32];                     /* attribute type */
      };
      longInt;                            /* attribute tag */

      /* Flags */
      boolean notForInput, useForInput;   /* use this pattern for
                                              input processing? */
      boolean notForOutput, useForOutput; /* use for output processing? */
      boolean notInSublist, useInSublist; /* include attr type in sublist? */
      boolean isNotAlias, isAlias;        /* mark attr value as an alias? */
```

```
      boolean isNotRecordRef, isRecordRef;/* reserved; use isNotRecordRef */

      align word;                         /* reserved */
      integer = $$CountOf(PatternArray);  /* pattern array size */
      array PatternArray {
         longint;                         /* pattern element type */
         integer;                         /* property number */
         integer;                         /* extra parameter */
      };
   };
};
```

IMPORTANT

The pattern element types in lookup tables are case sensitive. Thus, the following two patterns are *not* equivalent:

```
'word', prYear, 0;
'byte', prMonth, 0;
};
{
'Word', prYear, 0;
'Byte', prMonth, 0;
};
```

The Catalogs Extension would interpret the pattern element types `'word'` and `'byte'` as standard types, but would send `'Word'` and `'Byte'` to the code resource for processing. ▲

The flags field in each entry in the lookup table contains several bits that help select and control the translation process. Table 5-5 shows the currently defined flags (the rest are reserved for future use, and should be set to 0).

Table 5-5 Lookup-table flags

Flag	Meaning
`useForInput`	Use this table entry for translating attribute values to properties.
`useForOutput`	Use this table entry for translating properties to attribute values.
`useInSublist`	Include this attribute value in the sublist. This flag is used in aspect templates of records only. You must set this flag for each attribute type that you want included in the sublist of any information page that has a sublist and that uses this aspect. You should limit sublist items to one line. Multiline sublist items are not guaranteed to work correctly.
`isAlias`	The resulting entry in the sublist is an alias. This flag applies only to attributes in a sublist that hold aliases. If the `useInSublist` flag is not set, the CE ignores this flag.
`isRecordRef`	Reserved; use the `isNotRecordRef` value for this flag.

The Catalogs Extension uses the mapping of properties to attributes provided by the lookup table to check the user's access privileges for each attribute and to mark each property accordingly as either editable or uneditable. The CE can then use this information to allow or prevent a user from making changes in an information page. The CE assumes that properties not associated with any attribute are "internal" and therefore always editable (unless you explicitly make them uneditable with the `kDETcmdSetPropertyEditable` callback routine described on page 5-232).

When processing an output pattern (a lookup-table element that has the `useForOutput` flag set), the CE changes an existing attribute value if there is one or creates a new attribute value if one does not already exist. However, if there is no input pattern for that attribute type in the lookup table, the CE has no way of knowing an attribute value already exists, and so it creates a new one every time it processes the output pattern. In that case, the record ends up containing multiple attribute values corresponding to a single set of properties. Therefore, you must observe this rule:

IMPORTANT

You must always include an input pattern for every attribute type for which you provide an output pattern. ▲

If your lookup table or code resource writes a zero-length attribute value, the CE deletes that attribute value from the record. Note that an attribute type that contains an `RString` (for example) would not have zero-length attribute value unless the entire `RString` structure were removed; a zero-length `RString` still contains a length and a script code.

The input pattern can be separate from the output pattern, if you wish, and can be empty except for the resource declaration, the attribute type, the attribute tag, and the flags field.

Note

A lookup table can contain only one input pattern and one output pattern for each attribute type. Therefore, although the CE places no restriction on the number of attribute values that can be assigned to each attribute type, lookup-table patterns are designed to work only for those multivalued attributes that appear in sublists.

Multivalued attributes normally appear only in sublists, and the input and output patterns for an attribute in a sublist are normally located in the main aspect for that attribute type, not in the lookup table for the information page that contains the sublist. Therefore, you will not usually set the `useInSublist` flag and the `useForInput` or `useForOutput` flags for the same pattern element.

However, see "Conditional Element Types" on page 5-112 for pattern elements that you can use to develop exceptions to these rules. ◆

Basic Element Types

A number of pattern element types are available to process single pieces of the pattern. Table 5-6 shows the basic lookup-table element types available.

Table 5-6 Basic lookup-table element types

Element type	Data format	Property type
`'byte'`	Byte (8 bits)	Number
`'word'`	Word (16 bits)	Number
`'long'`	Long word (32 bits)	Number
`'pstr'`	Pascal-style string (8-bit length)	`RString`
`'wstr'`	Pascal-style string (16-bit length)	`RString`
`'cstr'`	C-style (null-terminated) string	`RString`
`'rstr'`	`RString` data structure	`RString`
`'type'`	4-character type	`RString`
`'blok'`	Block of data; the "extra parameter" field holds the number of bytes	Binary
`'bbit'`	Binary bit (8 per byte)	Number
`'abyt'`	Align to byte boundary	None
`'awrd'`	Align to word boundary	None
`'alng'`	Align to long boundary.	None
`'padz'`	Process the following element and pad it to the size specified in the extra field, using zero fill. (Used for fixed-width fields.)	None
`'rest'`	Take everything remaining in the attribute value and put it into a property.	Binary

Listing 5-9 shows the use of basic lookup-table elements to parse an attribute of type Album Track Info into several properties (the number of tracks and the hours, minutes, and seconds of playing time).

Listing 5-9 Lookup table with basic elements

```
// Properties
#define prNumTracks           kDETFirstDevProperty
#define prPlayingTimeHours    kDETFirstDevProperty + 1
#define prPlayingTimeMinutes  kDETFirstDevProperty + 2
#define prPlayingTimeSeconds  kDETFirstDevProperty + 3
```

```
// Lookup table
resource 'dett' (kMyAspectResourceID + kDETAspectLookup, purgeable) {
   {
   {"WAVE Album Track Info"}, typeBinary,
      useForInput, useForOutput, notInSublist, isNotAlias, isNotRecordRef,
      {
         'word', prNumTracks, 0;
         'long', prPlayingTimeHours, 0;
         'long', prPlayingTimeMinutes, 0;
         'long', prPlayingTimeSeconds, 0;
      };
   }
};
```

Conditional Element Types

Lookup tables also provide several element types that implement "if" statements. The element type indicates the test to be performed, the property number indicates which property to test, and the extra field of the element indicates a property number of the property against which the test is performed. If the condition in the test is met, the following element or block is executed. Otherwise, the following element or block is skipped. You can test against a constant value either by using one of the constant property numbers—allowing comparisons against constants in the range 0–249—or by using a resource-based static property. The conditional elements for lookup tables are shown in Table 5-7.

Table 5-7 Conditional elements for lookup tables

Element type	Pattern
`'equa'`	Value of the property specified by the "property number" field equal to value of the property specified by the "extra parameter" field (any type)
`'nteq'`	Value of the property specified by the "property number" field not equal to value of the property specified by the "extra parameter" field (any type)
`'less'`	Value of the property specified by the "property number" field less than value of the property specified by the "extra parameter" field (long integers only)

Table 5-7 Conditional elements for lookup tables (continued)

Element type	Pattern
`'grea'`	Value of the property specified by the "property number" field greater than value of the property specified by the "extra parameter" field (long integers only)
`'leeq'`	Value of the property specified by the "property number" field less than or equal to value of the property specified by the "extra parameter" field (long integers only)
`'greq'`	Value of the property specified by the "property number" field greater than or equal to value of the property specified by the "extra parameter" field (long integers only)

You can use the `'p:=p'` element type to set a property equal to the value of an existing property. The property field in a `'p:=p'` element indicates the destination property. The source property is given by the extra field. The source property (that is, the extra field) can specify a resource.

Listing 5-10 tests whether an album has a playing time of over 1 hour. If it does, the lookup table sets the property `prLongPlay` to `true`.

Listing 5-10 Lookup table with conditional elements

```
resource 'dett' (kMyAspectResourceID + kDETAspectLookup, purgeable) {
   {
   {"WAVE Album Track Info"}, typeBinary,
      useForInput, useForOutput, notInSublist, isNotAlias, isNotRecordRef,
      {
         'word', prNumTracks, 0;
         'long', prPlayingTimeHours, 0;
         'long', prPlayingTimeMinutes, 0;
         'long', prPlayingTimeSeconds, 0;
         'p:=p', prLongPlay, kDETFalseProperty;
         'greq', prPlayingTimeHours, kDETConstantProperty + 1;
         'p:=p', prLongPlay, kDETTrueProperty;
      };
   }
};
```

Block Elements

You can use block elements to form more complex patterns. You form a block by surrounding a list of elements with the `'(((('` and `')))))'` elements (Table 5-8). The block is then treated conceptually as a single element. Block elements are particularly

useful with repeating and conditional elements. In addition, if the destination property for the '((((' element is something other than kDETNoProperty, everything described by the block is put into the specified property as a binary block. (The destination property of the '))))' element must always be kDETNoProperty.)

Table 5-8 Block elements for lookup tables

Element type	Pattern
'(((('	Begin block
'))))'	End block

Listing 5-11 illustrates the use of the block elements ('((((' and '))))') with a conditional element. This lookup table tests whether an album has a playing time of over 1 hour. If it does, the lookup table sets the property prLongPlay to true and the property prComments to "Long-play album."

Listing 5-11 Lookup table with block elements

```
// Properties

#define prNumTracks           kDETFirstDevProperty
#define prPlayingTimeHours    kDETFirstDevProperty + 1
#define prPlayingTimeMinutes  kDETFirstDevProperty + 2
#define prPlayingTimeSeconds  kDETFirstDevProperty + 3
#define prComments            kDETFirstDevProperty + 4
#define prLongPlayComment     kDETFirstDevProperty + 5

resource 'rstr' (kMyAspectResourceID + prLongPlayComment, purgeable) {
   "Long-play album"
};

// Lookup table
resource 'dett' (kMyAspectResourceID + kDETAspectLookup, purgeable) {
   {
   {"WAVE Album Track Info"}, typeBinary,
      useForInput, useForOutput, notInSublist, isNotAlias, isNotRecordRef,
      {
         'word', prNumTracks, 0;
         'long', prPlayingTimeHours, 0;
         'long', prPlayingTimeMinutes, 0;
         'long', prPlayingTimeSeconds, 0;
         'p:=p', prLongPlay, kDETFalseProperty;
```

```
            'greq', prPlayingTimeHours, kDETConstantProperty + 1;
            '((((', kDETNoProperty, 0;
            'p:=p', prLongPlay, kDETTrueProperty;
            'p:=p', prComments, prLongPlayComment;
            '))))', kDETNoProperty, 0;
        };
    }
};
```

Listing 5-16 on page 5-131 illustrates the use of conditional and block elements in the implementation of a conditional view.

Size Element Types

Lookup tables provide pattern elements that create patterns of a specific size. Table 5-9 shows lookup-table elements you can use for this purpose. An attribute created by one of these pattern elements begins with a length (either a byte, word, or long integer, depending on the element), which is followed by data. The data is in the format specified by the next pattern element. You can use the block elements shown in Table 5-8 on page 5-114 to specify more complex formats for the data. If the data is longer than the pattern element used to format it, the pattern element is repeated as many times as necessary. You can use one of these elements to ensure that a particular attribute is of a specified size and format so that it can be read by a program external to the AOCE catalog system or by a CSAM.

A property created by one of these pattern elements is of property type Binary and of the size specified by the length byte, word, or long integer in the attribute.

Table 5-9 Lookup-table elements that create patterns of a specific size

Element type	Pattern	Property type
`'bsiz'`	Byte-sized size, followed by enough repeats of the following element to use that many bytes	Binary (property does not include size byte)
`'wsiz'`	Word-sized size, followed by enough repeats of the following element to use that many bytes	Binary (property does not include size word)
`'lsiz'`	Long word-sized size, followed by enough repeats of the following element to use that many bytes	Binary (property does not include size long word)

The code fragment in Listing 5-12 stores two properties that are variable-length text strings in a sized attribute. A CSAM or other program reading this attribute from the record could determine how many bytes to read from the length word without having to interpret the constituent `RString` structures.

Listing 5-12 Lookup table with size and block elements

```
// Properties
#define prAlbumName           kDETFirstDevProperty
#define prPublisherName       kDETFirstDevProperty + 1

// Lookup table
resource 'dett' (kMyAspectResourceID + kDETAspectLookup, purgeable) {
   {
   {"WAVE Album Name Info"}, typeBinary,
      useForInput, useForOutput, notInSublist, isNotAlias, isNotRecordRef,
      {
         'wsiz', kDETNoProperty, 0;
         '((((', kDETNoProperty, 0;
         'rstr', prAlbumName, 0;
         'rstr', prPublisherName, 0;
         '))))', kDETNoProperty, 0;
      };
   }
};
```

In this case, your information page template would assign the `prAlbumName` and `prPublisherName` properties to edit-text fields (see "View Lists" beginning on page 5-123). Your default value for each property should be a helpful text string such as "<enter text here>".

Suppose the user types the string "Apple's Top Hits" in the album name field and closes the information page. When it processes the lookup-table entry in Listing 5-12, the Catalogs Extension creates an attribute of type WAVE Album Name Info with the attribute tag `typeBinary`. On a roman script system (the script code is `smRoman` = 0), the attribute-value data looks like this:

```
0029
0000 0010 4170 706C 6527 7320 546F 7020 4869 7473    '....Apple's Top Hits'
0000 0011 3C65 6E74 6572 2074 6578 7420 6865 7265 3E '....<enter text here>'
```

The block of data begins with a word length (the length of the block of data, not including the length word itself), followed by two `RString` structures. Each `RString` begins with the script code (which is $0000 for roman script) followed by a length word followed by the string. Notice that the CE writes the value of the second string even

though it has not been changed, because the CE always fully specifies an attribute when any property derived from that attribute has changed.

The next time the user opens this information page, the CE looks through the lookup table for every attribute type for which the `useForInput` flag has been set. In this case, it finds the attribute type WAVE Album Name Info and finds one attribute value of this type, which has the attribute tag `typeBinary`. The CE finds the entry shown in Listing 5-12, which has this attribute type and tag type, and applies it to the data in the attribute value.

Because the lookup-table entry is of element type `'wsiz'`, the CE knows the first word of the data indicates the length of the rest of the data block. The rest of the lookup-table entry indicates that the data block consists of two `RString` structures, so the CE reads the length, script code, and string from the first `RString` structure and stores the value in the property `prAlbumName`. It then reads the second `RString` the same way, assigning the value to the property `prPublisherName`. The CE stores all strings internally as `RString` structures (see Table 5-2 on page 5-85).

You normally include a destination property for either a size element, for a block element, or for each element within the block, and make the destinations of all the other elements `kDETNoProperty`. To illustrate why this is so, Listing 5-13 shows a lookup-table entry similar to the one in Listing 5-12 except that, in Listing 5-13, the destination property for the `'wsiz'` element is `prNames` rather than `kDETNoProperty`.

Listing 5-13 Lookup-table entry with a destination property for the `'wsiz'` element type

```
// Properties
#define prAlbumName           kDETFirstDevProperty
#define prPublisherName       kDETFirstDevProperty + 1
#define prNames               kDETFirstDevProperty + 2

// Lookup table
resource 'dett' (kMyAspectResourceID + kDETAspectLookup, purgeable) {
   {
   {"WAVE Album Name Info"}, typeBinary,
      useForInput, useForOutput, notInSublist, isNotAlias, isNotRecordRef,
      {
         'wsiz', prNames, 0;
         '((((', kDETNoProperty, 0;
         'rstr', prAlbumName, 0;
         'rstr', prPublisherName, 0;
         '))))', kDETNoProperty, 0;
      };
   }
};
```

In the case of Listing 5-13, when the user types a new value into the album name or publisher name field and closes the information page, the CE first checks whether the binary property `prNames` already exists. If not, the CE uses the new value to create an attribute of type WAVE Album Name Info with the attribute tag `typeBinary`, exactly as it did for the lookup-table element in Listing 5-12 on page 5-116. If the property `prNames` exists, however (as it would if the code resource had already created it), the CE uses the value of the property `prNames` to create the attribute, ignoring the block following the `'wsiz'` element and therefore ignoring the new value typed in by the user.

In either case, the next time the user opens the information page, the CE creates the binary property `prNames` from the value of the attribute WAVE Album Name Info. After creating the property `prNames`, the CE uses the same attribute value to create the `RString` properties `prAlbumName` and `prPublisherName`, as it did for the lookup-table element in Listing 5-12 on page 5-116.

Because the property `prNames` exists, the CE ignores the block following the `'wsiz'` element and uses the value of the property `prNames` to create the attribute. Therefore, if the user changes the value of the property `prAlbumName` or `prPublisherName`, the value of the property `prNames` is not affected, and the changes to `prAlbumName` and `prPublisherName` are not saved. For this reason, you would normally include a destination property for either the size element, for the block element, or for each element within the block, and make the destinations of all the other elements `kDETNoProperty`. If you have some special need to maintain properties for both the block element or the size element and the constituents of the block, you must use your code resource to update property values.

Providing Your Own Pattern Elements

The Catalogs Extension passes to the aspect template's code resource any pattern element type that begins with an uppercase letter. The code resource can then process that portion of the attribute value or property. You can freely mix together built-in and code resource pattern elements.

When creating or changing attribute values, the CE processes only those properties that have changed and that are included in the list of properties in the lookup table. Because you can use a code resource routine invoked by a single pattern element to process any number of properties, you must use the following pattern element to list each property that your code resource processes:

Element type	Pattern
`'prop'`	Include this property in the lookup table

The `'prop'` pattern element lets you associate properties with a lookup table, so that the CE uses that lookup table when those properties need to be saved.

Overriding Default Property-Type Assignments

Each property has a type (see "Properties" beginning on page 5-84). The Catalogs Extension automatically converts between types as needed. The type of the property is taken from the pattern. In most cases, the type is determined automatically—`'byte'` pattern elements produce number type properties; `'rstr'` pattern elements produce string type properties; and so forth. Two pattern elements are provided to allow you to override the default type determination:

Element type	Pattern
`'styp'`	Set the type of the property to the value in the "extra parameter" field
`'btyp'`	Use the value of the "extra parameter" field as the type of all subsequent "binary" properties—for instance, `'((((', 'rest', 'bsiz', 'wsiz'` elements

You can use the `'styp'` and `'btyp'` element types to assign custom property types to properties. The CE calls your code resource when necessary to convert between your custom property types and standard property types; see "Custom Property-Type Conversions" beginning on page 5-188.

Canceling Pattern Processing

Lookup tables provide an element type that aborts processing of the pattern.

Element type	Pattern
`'abrt'`	Abort processing of the pattern

The `'abrt'` lookup-table element stops the processing of the pattern but does not abort the process of reading or writing the attribute value.

Components of Information Page Templates

Information page templates specify the layout and provide the functions of the information pages that users see when they open a record or attribute. The primary content of an information page template is one or more view lists, indicating where on the information page to place the fields (or *views*) that display information to users and allow them to edit that information.

The Catalogs Extension fills in the views from an aspect specified by the information page. A view list specifies only the property numbers of properties in that aspect. All of the information in the main part of an information page (that is, all of the information page except for items in a sublist) comes from the same, specified aspect.

An information page template can contain the resources listed in Table 5-10.

Table 5-10 Resources in information page templates

Resource type	Offset of resource ID from signature resource ID	Purpose of resource
'deti'	0	Identifies template as information page and provides a base resource ID. Required for all information page templates.
'rstr'	kDETTemplateName	Name of template. Required for all information page templates.
'rstr'	kDETRecordType	Type of record to which the template applies. Either this resource, the kDETAttributeType resource, or both must be included.
'rstr'	kDETAttributeType	Type of attribute to which the template applies. Either this resource, the kDETRecordType resource, or both must be included.
'detn'	kDETAttributeValueTag	Attribute tag of attributes to which the template applies. You can provide this resource if you have also provided the kDETAttributeType resource. If you don't provide this resource, the template applies to attributes with any tag value.
'detv'	The resource ID of a view list is independent of that of the information page signature resource.	View list. An information page template can contain any number of view lists describing specific items in the information page and in the sublist (if any). The information page signature resource lists the resource IDs of all of the applicable view list resources. Each information page must have at least one view list if it is to provide any useful information.
'rstr'	kDETInfoPageMainViewAspect	The name of the aspect template whose aspect provides all of the properties for the views in the main portion of the information page. This aspect also lists the objects that go in the sublist (if any). Required for every information page template.
'rstr'	kDETInfoPageName	The name of the information page that appears in the page-selection pop-up menu. Required for all information pages that appear in a page-selection pop-up menu.
'detm'	kDETInfoPageMenuEntries	A list of items that the CE adds to the end of the Catalogs menu. When the user chooses one of these items, the CE calls the code resource of the aspect. Optional.

Information Page Template Signature Resource

The information page seen by the user is the combination of one or more view lists specified by the template. The signature resource of the information page lists the resource IDs of the view lists to be used in the main and icon-list parts of the information page. Each view list resource ID is preceded by two property numbers. The view list is displayed only if the contents of the two properties are equal or if either property equals `kDETNoProperty`. This feature allows you to enable or disable a view list according to the current values of properties in the associated aspect. Thus, an information page can change to include different views according to the current state of the record or attribute represented by that page.

The signature resource also specifies the sort-order number of the information page, the presence or absence of a sublist in the information page, and the rectangle that contains the sublist. The Catalogs Extension displays the information pages in the sequence indicated by their sort-order numbers. When creating several information pages for a single catalog object, you should assign sort-order numbers at intervals of 1000 (1000, 2000, 3000, and so on) to allow a new page to be easily inserted between two existing pages.

The signature resource includes a Boolean value that indicates whether the first edit-text field of the information page should be automatically selected when the information page is opened. If you set this value to `selectFirstText`, the CE selects the first edit-text field when a user opens the information page (as is usual for a dialog box). If you set this value to `noSelectFirstText`, no field is initially selected. For most information pages, you should set the Boolean value to `noSelectFirstText` to lessen the likelihood that users will change a field unintentionally.

'deti' Resource

The signature resource for an information page template is of type `'deti'`.

```
type 'deti' {
   longInt = kDETInfoPageVersion;   /* template format version */
   longInt;                         /* sort order */
   rect;                            /* rectangle to put sublist in */
   boolean selectFirstText, noSelectFirstText;
                                 /* select the first text field
                                     when info-page opens? */
   align word;                   /* reserved */
   integer = $$CountOf(HeaderViewArray);
   array HeaderViewArray {       /* the header view lists */
      integer;                   /* property 1 */
      integer;                   /* property 2 */
      integer;                   /* 'detv' ID */
```

```
   };
   integer = $$CountOf(SubviewViewArray);
   array SubviewViewArray {      /* the subview view lists */
      integer;                   /* property 1 */
      integer;                   /* property 2 */
      integer;                   /* 'detv' ID */
   };
};
```

Listing 5-14 is an example of an information page template signature resource. This resource specifies four view lists. The first describes a view that always appears on the information page. The second and third are conditional views; each appears on the information page only if the two properties associated with that view are equal (or if either property number equals `kDETNoProperty`). In Listing 5-14, the view list with an ID of `rMyInfoPage` + 1 appears if the property `prEnable1` equals `kDETZeroProperty` (that is, 0) and the view list `rMyInfoPage` + 2 appears if `rEnable1` equals `kDETOneProperty` (that is, 1). The fourth view list in Listing 5-14 describes a line in the sublist and is always enabled.

Listing 5-14 Information page signature resource with conditional views

```
resource 'deti' (rMyInfoPage, purgeable) {
   1000,                                          // sort order
   kDETSublistRect,                               // rect for sublist
   noSelectFirstText,
   {
   // View list for main part of window:
   kDETNoProperty, kDETNoProperty, rMyInfoPage;    // always enabled

   // View lists for conditional views in main part of window
   prEnable1, kDETZeroProperty, rMyInfoPage + 1;   // enabled if prEnable1
                                                   //  property is 0
   prEnable1, kDETOneProperty, rMyInfoPage + 2;    // enabled if prEnable1
                                                   //  property is 1
   },
   // View lists for sublist:
   {
   kDETNoProperty, kDETNoProperty, rMyInfoPage + 3;// always enabled
   }
};
```

Listing 5-16 on page 5-131 further illustrates the implementation of conditional views.

If the information page has no sublist, you can use the following sublist rectangle:

```
#define kDETNoSublistRect  {0, 0, 0, 0}
```

View Lists

A view list specifies individual items on the information page. Each item in the list includes the graphic rectangle containing the view, the number of the property that provides the information to be displayed, the type of view, and information specific to that view type. A view list need not use all of the properties in the aspect. On the other hand, a single property can provide information for more than one view. For example, a set of three radio buttons would require three views in a view list but could all display the information in a single property.

'detv' Resource

A view list is defined by the `'detv'` resource type.

```
type 'detv' {
   longint = 0;
   longint = 0;
   longint = 0;
   integer = 0;
   longint = 0;
   longint = 0;
   longint = 0;
   longint = 0;
   integer = 0;
   longint = 0;
   longint = 0;

   integer = $$CountOf(ItemArray);/* count */
   array ItemArray {
      rect;                  /* bounds of the view */
      longint = 0;           /* position flags (preset by CE) */
      longint;               /* flags (described following this struct) */
      integer;               /* property number */
      switch {               /* class of view */

         case StaticTextFromView:
            key longint = 7750;/* class ID */
            integer;         /* font ID */
            integer;         /* font size */
```

```
   integer;          /* justification */
   integer;          /* style */
   pstring;          /* string to display */

case StaticCommandTextFromView:
   key longint = 22250;/* class ID */
   integer;          /* font ID */
   integer;          /* font size */
   integer;          /* justification */
   integer;          /* style */
   pstring;          /* string to display */
   align word;       /* reserved */
   longint;          /* property command */
   integer;          /* command parameter */

case StaticText:
   key longint = 3250;/* class ID */
   integer;          /* font ID */
   integer;          /* font size */
   integer;          /* justification */
   integer;          /* style */

case EditText:
   key longint = 8250;/* class ID */
   integer;          /* font ID */
   integer;          /* font size */
   integer;          /* justification */
   integer;          /* style */

case Bitmap:
   key longint = 6250;
   integer;          /* size */

case Box:
   key longint = 4750;/* class ID */
   integer;          /* box attributes */

case DefaultButton:
   key longint = 7250;/* class ID */
   integer;          /* font ID */
   integer;          /* font size */
   integer;          /* justification */
   integer;          /* style */
```

```
   pstring;          /* label for button */
   align word;       /* reserved */
   longint;          /* property command */

case Button:
   key longint = 21000;/* class ID */
   integer;          /* font ID */
   integer;          /* font size */
   integer;          /* justification */
   integer;          /* style */
   pstring;          /* label for button */
   align word;       /* reserved */
   longint;          /* property command */

case CheckBox:
   key longint = 21250;/* class ID */
   integer;          /* font ID */
   integer;          /* font size */
   integer;          /* justification */
   integer;          /* style */
   pstring;          /* label for checkbox */
   align word;       /* reserved */
   longint;          /* property command */

case RadioButton:
   key longint = 21500;/* class ID */
   integer;          /* font ID */
   integer;          /* font size */
   integer;          /* justification */
   integer;          /* style */
   pstring;          /* label for button */
   align word;       /* reserved */
   longint;          /* property command */
   integer;          /* command parameter */

case Menu:
   key longint = 5750;/* class ID */
   integer;          /* font ID */
   integer;          /* font size */
   integer;          /* justification */
   integer;          /* style */
   pstring;          /* label for pop-up menu */
   align word;       /* reserved */
```

```
            longint;          /* property command */
            integer;          /* menu resource ID */

         case EditPicture:
            key longint = 0x00010000 + 24250; /* class ID */
            integer;          /* maximum pixel depth */

         case Custom:
            key longint = 6750;/* class ID */
            integer;          /* reference value for use by developer */
      };
      align word;
   };
};
```

Here are the possible values for the flags field near the beginning of the `'detv'` resource:

Flag	Meaning
`kDETNoFlags`	No flags.
`kDETEnabled`	If set to 1, this view should be highlighted if the user selects the item of which this view is a part. Not for use in sublists.
`kDETHilightIfSelected`	If set to 1, highlight the view when the entry is selected. For items in a sublist only.
`kDETNumericOnly`	If set to 1, the user is allowed to enter only numbers into this item. For editable text fields only.
`kDETMultiLine`	If set to 1, the user or template can enter more than one line into the field. If set to 0, pressing Return terminates text entry; if set to 1, pressing Return enters a carriage return and starts a new line. For text fields only. Note that you must set this flag to 1 for multiline static text fields.
`kDETDynamicSize`	If set to 1, a box appears around the item when the user clicks the item or tabs into it. The CE sizes the box dynamically as the user edits the text in the box. For editable text fields only.
`kDETAllowNoColons`	If set to 1, the user is not allowed to enter colons as part of the text. The CE substitutes a hyphen (-) for each colon (:) typed. For editable text fields in which the user is expected to type a filename.
`kDETPopupDynamicSize`	If set to 1, the CE automatically resizes a pop-up menu according to its contents. For pop-up menus only.
`kDETScaleToView`	If set to 1, a picture is scaled to the bounds of the view. If set to 0, the picture is cropped. For `EditPicture` views only.

Several constants are provided for use with fonts:

Constant	Value
kDETApplicationFont	1
kDETApplicationFontSize	9
kDETAppFontLineHeight	12
kDETSystemFont	0
kDETSystemFontSize	12
kDETSystemFontLineHeight	16
kDETDefaultFont	1
kDETDefaultFontSize	9
kDETDefaultFontLineHeight	12

Here are the possible values for text styles:

Constant	Meaning
kDETNormal	Normal font
kDETBold	Bold
kDETItalic	Italic
kDETUnderline	Underlined
kDETOutline	Outline style
kDETShadow	Shadow style
kDETCondense	Condensed
kDETExtend	Extended
kDETIconStyle	Same as normal text style for regular entries and as italic text style for aliases

Here are the possible values for text justification:

Value	Meaning
kDETLeft	Text in scripts written from left to right is left-justified; text in scripts written from right to left is right-justified.
kDETCenter	All text is centered in the text field.
kDETRight	All text is right-justified.
kDETForceLeft	All text is left-justified.

View types

`StaticTextFromView`

Text that cannot be edited by the user. The contents of the string come from the view itself; that is, from the information page template.

`StaticCommandTextFromView`
: Text that cannot be edited by the user. When the user clicks on the text, the CE calls your code resource with the routine selector `kDETcmdPropertyCommand` and with the property command and command parameter from the view list. Most commonly, the code resource does nothing in response to this property command; instead, this view type is used to provide headings for sublist columns, so that when the user clicks the column heading, the sort criteria for the sublist is changed. To accomplish this, set the property-command field to `kDETChangeViewCommand` and the command parameter field to the negative of the property number of the appropriate entry in the aspect's `kDETAspectViewMenu` resource (page 5-103). When you use `kDETChangeViewCommand` as the property command, the CE handles the command without calling your code resource. The contents of the string come from the "string to display" field in the view itself.

`StaticText`
: Text that cannot be edited by the user. The contents of the string come from the view property; that is, from the aspect.

`EditText`
: Text that the user can edit. The contents of the string come from the view property; that is, from the aspect.

`Bitmap`
: An icon, taken from an icon suite in the aspect template. The CE looks for an icon suite with a resource ID equal to the aspect template's base ID plus the property number. The property number is in the property number field of the `'detv'` structure. The size field indicates the size of the icon: `kDETLargeIcon`, `kDETSmallIcon`, or `kDETMiniIcon`.

`Box`
: A simple graphic rectangle or rounded rectangle, useful for drawing dividing lines between view elements and boxes around elements that do not normally have them, such as sublists. The `Box` view type takes a single integer as a parameter. The dimensions of the box are those of the view itself. The first 4 bytes of the integer are flags, as follows:

Bit	Flag	Meaning
none	`kDETUnused`	No flags.
0	`kDETBoxTakesContentClicks`	If this flag is set to 1 and the user clicks in the box, the CE calls your code resource with the property number.
1	`kDETBoxIsRounded`	Box is a rectangle with rounded corners.
2	`kDETBoxIsGrayed`	Box is dimmed.
3	`kDETBoxIsInvisible`	Box is invisible.

`DefaultButton` A standard button with a heavy border indicating that this is the default button; that is, pressing Return or Enter keys has the same effect as clicking the button. Clicking a default button closes any open edit-text field, whereas clicking a regular button does not. In every other respect, a `DefaultButton` view is identical to a `Button` view. You can have only one default button in a given information page.

`Button` A standard button. You must use the button's property number for the property-command field of the `'detv'` resource. Then, when the user clicks the button, the CE calls your code resource with the routine selector `kDETcmdPropertyCommand` and with the property number of the button in the `property` field of the parameter block. If your code resource does not handle this command, the CE does nothing. If you use the property numbers `kDETAddNewItem`, `kDETRemoveSelectedItems`, or `kDETOpenSelectedItems`, the CE handles the command without calling your code resource. These property numbers are described in Table 5-3 on page 5-86.

`Checkbox` A standard dialog checkbox, which can be selected or not. The property can be equal to 0 (checkbox is not selected) or 1 (checkbox is selected). You must use the checkbox's property number for the property-command field of the `'detv'` resource. Then when the user clicks the checkbox, the CE sends the property number to your code resource. If your code resource does not handle the command, the CE changes the property value to toggle the checkbox off or on.

Note that if your code resource does handle this command, you must call the `kDETcmdDirtyProperty` callback routine (page 5-233) to force the CE to redraw the checkbox.

`RadioButton` A standard dialog radio button, which can be on or off, as selected by the user. When multiple radio buttons are associated with the same property, only one can be on at a time. You must use the button's property number for the property-command field of the `'detv'` resource, and you should use a different value for the command parameter field of each button associated with the same property. Set the command parameter of the default button (the one you want the CE to select when the view is first displayed) equal to the value of the property. When the user clicks the radio button, the CE first calls your code resource. If your code resource does not handle the command, the CE sets the value of the property to the value of the parameter for that button, thereby selecting that button and deselecting all other radio buttons for that property.

Note that if your code resource does handle this command, you must call the `kDETcmdDirtyProperty` callback routine (page 5-233) to force the CE to redraw the radio button.

`Menu` A pop-up menu from which the user can select one item, which the information page then displays as the "state" of the menu. The menu resource ID field of the `'detv'` resource indicates the `'fmnu'` resource that specifies the contents of the menu. The menu resource can have any resource ID. You must use the menu's

property number for the property-command field of the `'detv'` resource. Pop-up menus are limited to 31 items; if your `'fmnu'` resource includes more than 31 items, the pop-up menu will not work properly. You cannot put a pop-up menu view in a sublist.

When the user chooses an item in the pop-up menu, the CE calls your code resource with the routine selector `kDETcmdPropertyCommand` and with the property number of the menu in the `property` field of the parameter block. The CE gets the value for the `parameter` field of the command's parameter block from the `'fmnu'` resource; your code resource can use this parameter to determine which item in the pop-up menu the user has selected. You can use the `kDETcmdAddMenu` (page 5-238) and `kDETcmdRemoveMenu` (page 5-240) callback routines to add and remove menu items.

`EditPicture` A picture that the user can select and copy onto the Clipboard, cut, or replace by pasting.

`Custom` A custom view defined by the code resource of the aspect associated with the information page. A custom view can respond to mouse-down events if you provide code to handle these events. There is no way for the user to select the custom view, but if no other view is selected, your code resource can receive keypress events and interpret them as belonging to the custom view. See "Custom Views and Custom Menus" beginning on page 5-192 for more information about how code resources can handle custom views. You can specify any value you wish for the reference-value integer in the view list. Your code resource can use the `kDETcmdGetCustomViewUserReference` callback routine (page 5-242) to obtain this value.

Listing 5-15 shows a sample view list.

Listing 5-15 Sample view list

```
resource 'detv' (rMyInfoPage, purgeable) {
   {
   {kBitmapTop, kBitmapLeft, kBitmapBottom, kBitmapRight},
   kDETNoFlags, kDETAspectMainBitmap,
   Bitmap { kDETLargeIcon };

   {kStatTextTop, kStatTextLeft, kStatTextBottom, kStatTextRight},
   kDETNoFlags, kDETNoProperty,
   StaticTextFromView { kDETApplicationFont, kDETApplicationFontSize,
                        kDETRight, kDETBold, "Label:" };

   {kEditTextTop, kEditTextLeft, kEditTextBottom, kEditTextRight},
   kDETEnabled, prEditTextProperty,
   EditText { kDETApplicationFont, kDETApplicationFontSize, kDETLeft,
```

```
            kDETNormal};

   {kCheckboxTop, kCheckboxLeft, kCheckboxBottom, kCheckboxRight},
   kDETNoFlags, prCheckboxProperty,
   CheckBox { kDETApplicationFont, kDETApplicationFontSize,
            kDETLeft, kDETNormal, "Check Me", prCheckboxProperty };

   {kRadio1Top, kRadio1Left, kRadio1Bottom, kRadio1Right},
   kDETNoFlags, prRadioProperty,
   RadioButton { kDETApplicationFont, kDETApplicationFontSize,
               kDETLeft, kDETNormal, "Radio 1", prRadioProperty, 0 };

   {kRadio2Top, kRadio2Left, kRadio2Bottom, kRadio2Right},
   kDETNoFlags, prRadioProperty,
   RadioButton { kDETApplicationFont, kDETApplicationFontSize,
               kDETLeft, kDETNormal, "Radio 2", prRadioProperty, 1 };
   }
};
```

Implementing Conditional Views

Listing 5-16 shows a lookup table, information page signature resource, and view list used to implement a conditional view. In Listing 5-16, the information page contains a view-selection pop-up menu with three choices. When the user chooses an item from the menu, the value of that item becomes the value of the property `prMsgType`. In the `'deti'` resource, the value of `prMsgType` determines which view is active. In the lookup table (the `'dett'` resource), the value of `prMsgType` determines which properties are processed. Note the use of block and conditional elements in the lookup table (see "Conditional Element Types" on page 5-112 and "Block Elements" on page 5-113) to achieve this end. There is one view list for the pop-up menu and one view list for each of the conditional views.

Note
Listing 5-16 is not a complete, working set of templates. It is intended only to illustrate the interaction of the information page signature resource, lookup table, and view lists in the implementation of conditional views. ◆

Listing 5-16 Implementing a conditional view

```
#define prMsgType       kDETFirstDevProperty
#define prPMDate        kDETFirstDevProperty + 1
#define prPMTime        kDETFirstDevProperty + 2
#define prPMFrom        kDETFirstDevProperty + 3
```

```
#define prPMTo          kDETFirstDevProperty + 4
#define prNMessage      kDETFirstDevProperty + 5
#define prNReply        kDETFirstDevProperty + 6
#define prIBOFrom       kDETFirstDevProperty + 7
#define prIBOTo         kDETFirstDevProperty + 8
#define prIBOBecause    kDETFirstDevProperty + 9
resource 'deta' (kCondViewAspect, purgeable) {
   0,                // drop-operation order
   dropCheckAlways,  // drop-check flag
   notMainAspect     // not the main aspect
};

resource 'rstr' (kCondViewAspect + kDETTemplateName, purgeable) {
   "WAVE Conditional view aspect"   // Start with application signature
};

resource 'rstr' (kCondViewAspect + kDETRecordType, purgeable) {
   "WAVE Conditional View"          // Start with application signature
};

// Custom information page window with no default pop-up menu
resource 'detw' (kCondViewAspect + kDETAspectInfoPageCustomWindow, purgeable)
   {
   {0,0,224,224},
   discludePopup
   };

// Lookup table. Conditional elements correspond to conditional views.
resource 'dett' (kCondViewAspect + kDETAspectLookup, purgeable) {
   {
   {"aoce MailNote"}, typeBinary,
      useForInput, useForOutput, notInSublist, isNotAlias, isNotRecordRef,
      {
         'long', prMsgType, 0;
         'equa', prMsgType, kDETConstantProperty + 0;
            '(((((', kDETNoProperty, 0;
            'rstr', prPMDate, 0;
            'rstr', prPMTime, 0;
            '))))', kDETNoProperty, 0;
         'equa', prMsgType, kDETConstantProperty + 1;
            '(((((', kDETNoProperty, 0;
            'rstr', prNMessage, 0;
            'rstr', prNReply, 0;
```

```
            '))))', kDETNoProperty, 0;
         'equa', prMsgType, kDETConstantProperty + 2;
            '((((', kDETNoProperty, 0;
            'rstr', prIBOFrom, 0;
            'rstr', prIBOTo, 0;
            'rstr', prIBOBecause, 0;
            '))))', kDETNoProperty, 0;
      };
   }
};

resource 'deti' (kCondViewInfoPage, purgeable) {
   1000,
   kDETNoSublistRect,
   noSelectFirstText,
   {
   kDETNoProperty, kDETNoProperty, kCondViewInfoPage;
   prMsgType, kDETConstantProperty + 0, kCondViewInfoPage + 1;
   prMsgType, kDETConstantProperty + 1, kCondViewInfoPage + 2;
   prMsgType, kDETConstantProperty + 2, kCondViewInfoPage + 3;
   },
   {
   }};

resource 'rstr' (kCondViewInfoPage + kDETTemplateName, purgeable) {
   "WAVE Conditional view info page"
};

resource 'rstr' (kCondViewInfoPage + kDETRecordType, purgeable) {
   "WAVE Conditional view"
};

resource 'rstr' (kMNInfoPage + kDETInfoPageMainViewAspect, purgeable) {
   "WAVE Conditional view aspect"
};

//View list for conditional-view-selection pop-up menu
resource 'detv' (kMNInfoPage, purgeable) {
   {
   kMenuTop, kMenuLeft, kMenuBottom, kMenuRight,
   kDETNoFlags, prMsgType,
   Menu {kDETSystemFont, kDETSystemFontSize, kDETCenter, kDETNormal, "",
         prMsgType, kMNInfoPage};
```

```
   }
}

//Menu resource for conditional-view-selection pop-up menu
resource 'fmnu' (kMNInfoPage, purgeable)
   {
   kMNInfoPage,
   {
   0,”Phone Message”;
   1,”Note”;
   2,”I’ll Be Out”;
   }
   };

//View lists for conditional views
resource 'detv' (kMNInfoPage + 1, purgeable) {
   {
   {kTextTop, kLabelLeft, kTextTop + kOneLineHeight, kLabelRight},
   kDETNoFlags, kDETNoProperty,
   StaticTextFromView { kLabelFont, kLabelSize, kDETRight, kLabelStyle,
   "Date:" };

   {kTextTop - 2, kTextColumnLeft, kTextTop + kOneLineHeight - 2,
   kTextRight},
   kDETEnabled, prPMDate,
   EditText { kTextFont, kTextSize, kDETLeft, kTextStyle };

   {kTextTop + kOneLineHeight + 2, kLabelLeft,
   kTextTop + kOneLineHeight + 2 + kOneLineHeight, kLabelRight},
   kDETNoFlags, kDETNoProperty,
   StaticTextFromView { kLabelFont, kLabelSize, kDETRight, kLabelStyle,
   "Time:" };

   {kTextTop + kOneLineHeight + 2 - 2, kTextColumnLeft,
   kTextTop + kOneLineHeight + 2 + kOneLineHeight - 2, kTextRight},
   kDETEnabled, prPMTime,
   EditText { kTextFont, kTextSize, kDETLeft, kTextStyle };
   };
};

resource 'detv' (kMNInfoPage + 2, purgeable) {
   {
   {kTextTop - 2, kTextLeft, kTextTop + kSixLineHeight - 2, kTextRight},
```

```
kDETMultiLine, rNMessage,
   EditText { kTextFont, kTextSize, kDETLeft, kTextStyle };

   {kTextTop + kSixLineHeight + 2, kLabelLeft, kTextTop + kSixLineHeight + 2
+ kOneLineHeight, kLabelRight}, kDETNoFlags, kDETNoProperty,
   StaticTextFromView { kLabelFont, kLabelSize, kDETRight, kLabelStyle,
"Reply:" };

   {kTextTop + kSixLineHeight + 2 + kOneLineHeight + 2 -6, kTextLeft,
kTextBottom, kTextRight}, kDETMultiLine, rNReply,
   EditText { kTextFont, kTextSize, kDETLeft, kTextStyle };
   };
};

resource 'detv' (kMNInfoPage + 3, purgeable) {
   {
   {kTextTop, kLabelLeft, kTextTop + kOneLineHeight, kLabelRight},
kDETNoFlags, kDETNoProperty,
   StaticTextFromView { kLabelFont, kLabelSize, kDETRight, kLabelStyle,
"From:" };

   {kTextTop - 2, kTextColumnLeft, kTextTop + kOneLineHeight - 2,
kTextRight}, kDETEnabled, rIBOFrom,
   EditText { kTextFont, kTextSize, kDETLeft, kTextStyle };

   {kTextTop + kOneLineHeight + 2, kLabelLeft, kTextTop + kOneLineHeight + 2
+ kOneLineHeight, kLabelRight}, kDETNoFlags, kDETNoProperty,
   StaticTextFromView { kLabelFont, kLabelSize, kDETRight, kLabelStyle,
"To:" };

   {kTextTop + kOneLineHeight + 2 - 2, kTextColumnLeft, kTextTop +
kOneLineHeight + 2 + kOneLineHeight - 2, kTextRight}, kDETEnabled, rIBOTo,
   EditText { kTextFont, kTextSize, kDETLeft, kTextStyle };

   {kTextTop + kOneLineHeight + 2 + kOneLineHeight + 2, kLabelLeft, kTextTop
+ kOneLineHeight + 2 + kOneLineHeight + 2 + kOneLineHeight, kLabelRight},
kDETNoFlags, kDETNoProperty,
   StaticTextFromView { kLabelFont, kLabelSize, kDETRight, kLabelStyle,
"Because:" };

   {kTextTop + kOneLineHeight + 2 + kOneLineHeight + 2 - 2, kTextColumnLeft,
kTextTop + kOneLineHeight + 2 + kOneLineHeight + 2 + kTwoLineHeight - 2,
kTextRight}, kDETEnabled, rIBOBecause,
```

```
   EditText { kTextFont, kTextSize, kDETLeft, kTextStyle };
   };
};
```

Sublists

If the information page includes a sublist, the information page template includes a view list that describes a line in the list. Each line in a sublist typically contains an icon, a "name" field, and a "kind" field, and might contain other fields or controls. For each item in the sublist, the Catalogs Extension takes the properties for the views from the main aspect of that item.

Note that, if the sublist contains items of more than one type, each type of item has an associated main aspect template. For example, if the sublist contains Direct Dialup mail addresses, PowerTalk serverless mail addresses, and PowerShare server mail addresses, there are three main aspect templates, one for each type of mail address. The CE uses the appropriate main aspect template to create the main aspect for each item that appears in the list. The CE uses a single view list in the information page template to format all of the lines in the sublist but takes the values to display in the sublist from a separate main aspect for each line.

The CE does not automatically draw a box around a sublist; if you want a box around a sublist, you must draw it yourself.

View lists are described in the preceding section. Main aspect templates are described in "Main Aspect Template Resources" beginning on page 5-88. Listing 5-4 on page 5-44 shows an information page template for an information page with a sublist.

Information Page Resources

In addition to view lists, an information page template contains resources that name the template, specify the type of record or attribute to which the template applies, provide the name of the associated aspect, and specify items for the Catalogs menu. The information page template resources that are common to aspect templates are described in "Template Names" on page 5-75 and "Specifying Record and Attribute Types for Templates" on page 5-75. The remaining resources are described in this section. For a complete list of information page resources, see Table 5-10 on page 5-120.

kDETInfoPageMainViewAspect

All property numbers listed by the view lists for the main portion of the information page come from one aspect, the *main view aspect*. The main view aspect also provides the list of objects to be included in the sublist (if any). Name the main view aspect with an `RString` resource that has a resource ID with an offset of `kDETInfoPageMainViewAspect` from the template's base resource.

Note
Do not confuse the *main view aspect* with a *main aspect*. A main aspect provides the properties that describe an item in a sublist. A main view aspect provides all of the properties needed by the information page *except* the contents of the sublist. ◆

```
resource 'rstr' (rMyInfoPage + kDETInfoPageMainViewAspect, purgeable) {
   "WAVE Associated Aspect Name"
};
```

IMPORTANT
If the information page template does not name a main view aspect, the Catalogs Extension does not load the template. ▲

kDETInfoPageName

To specify the name of the information page that appears in the page-selection pop-up menu, use an `RString` resource that has a resource ID with an offset of `kDETInfoPageName` from the template's base resource.

```
resource 'rstr' (rMyInfoPage + kDETInfoPageName, purgeable) {
   "Information Page Pop-up Name"
};
```

You must include this resource in every information page template if the information page window includes a page-selection pop-up menu. Only custom information pages can exclude page-selection pop-up menus; see the description of the `kDETAspectInfoPageCustomWindow` resource on page 5-97.

kDETInfoPageMenuEntries

You can use a resource of type `'detm'` to add items to the Catalogs menu. Give the resource an ID with an offset of `kDETInfoPageMenuEntries` from the template's base resource ID.

```
resource 'detm' (rMyInfoPage + kDETInfoPageMenuEntries, purgeable) {
   rMyInfoPage + kDETInfoPageMenuEntries,
   {
      menuParameter1, "Entry 1";
      menuParameter2, "Entry 2";
   }
};
```

When the user chooses an item, the Catalogs Extension calls the code resource in the aspect with the routine selector `kDETcmdCustomMenuSelected` (page 5-195), passing your code the menu parameter from the `kDETInfoPageMenuEntries` resource for the item the user selected. Your code resource is also called (with the `kDETcmdCustomMenuEnabled` routine selector described on page 5-194) to determine if the menu item should be enabled.

Components of Forwarder Templates

Forwarder templates provide a list of names of aspect and information page templates that can be used with the record or attribute type to which the template applies.

A forwarder template can contain the resources listed in Table 5-11.

Table 5-11 Resources in forwarder templates

Resource type	Offset of resource ID from signature resource ID	Purpose of resource
`'detf'`	0	Identifies template as forwarder and provides a base resource ID. Required for all forwarder templates.
`'rstr'`	`kDETTemplateName`	Name of template. Required for all forwarder templates.
`'rstr'`	`kDETRecordType`	Type of record to which the template applies. Either this resource, the `kDETAttributeType` resource, or both must be included.
`'rstr'`	`kDETAttributeType`	Type of attribute to which the template applies. Either this resource, the `kDETRecordType` resource, or both must be included.
`'detn'`	`kDETAttributeValueTag`	Attribute tag of attributes to which the template applies. You can provide this resource if you have also provided the `kDETAttributeType` resource. If you don't provide this resource the template applies to attributes with any tag value.
`'rst#'`	`kDETForwarderTemplateNames`	A list of names of aspect and information page templates that the CE can use with the record or attribute type to which this forwarder template applies.

Forwarder Template Signature Resource

The forwarder template signature resource provides a base resource ID from which the other resource IDs in the template are offset and provides the forwarder template version number of the template.

'detf' Resource

Use a resource of type `'detf'` for the forwarder template signature resource.

```
type 'detf' {
   longInt = kDETForwarderVersion;  /* template format version */
};
```

Forwarder Template Resources

A forwarder template contains resources that name the template, specify the type of record or attribute to which the template applies, and specify the names of templates that the Catalogs Extension can use with the record or attribute to which the template applies. The forwarder template resources that are common to aspect and information page templates are described in "Template Names" on page 5-75 and "Specifying Record and Attribute Types for Templates" on page 5-75. The remaining resource is described in this section. For a complete list of forwarder template resources, see Table 5-11 on page 5-138.

kDETForwarderTemplateNames

To specify the templates that the Catalogs Extension can use with the record or attribute to which the forwarder template applies, use a resource of type `'rst#'` that has a resource ID with an offset of `kDETForwarderTemplateNames` from the template's base resource.

```
resource 'rst#' (kForwarderTemplate + kDETForwarderTemplateNames, purgeable)
{
   { "WAVE Album", "WAVE Track" }
};
```

Components of Killer Templates

A killer template provides a list of names of templates that the Catalogs Extension should ignore. You can use killer templates to disable any type of template except another killer template. A killer template can contain the resources listed in Table 5-12.

Table 5-12 Resources in killer templates

Resource type	Offset of resource ID from signature resource ID	Purpose of resource
`'detk'`	0	Identifies template as killer and provides a base resource ID. Required for all killer templates.
`'rstr'`	`kDETTemplateName`	Name of template. Required for all killer templates.
`'rst#'`	`kDETKillerName`	A list of names of templates to disable.

Killer Template Signature Resource

The killer template signature resource provides a base resource ID from which the other resource IDs in the template are offset and provides the killer template version number of the template.

'detk' Resource

Use a resource of type `'detk'` for the killer template signature resource.

```
type 'detk' {
   longInt = kDETKillerVersion;  /* template format version */
};
```

Killer Template Resources

A killer template contains resources that name the template and specify the names of templates that the Catalogs Extension should ignore. The killer template name resource is described in "Template Names" on page 5-75. The other resource is described in this section. For a complete list of killer template resources, see Table 5-12.

kDETKillerName

To specify the templates that the CE should disable, use a resource of type `'rst#'` that has a resource ID with an offset of `kDETKillerName` from the template's base resource.

```
resource 'rst#' (kKillerTemplate + kDETKillerName, purgeable) {
   { "WAVE Album", "WAVE Track" }
};
```

Components of File Type Templates

A file type template provides a list of file types that the Catalogs Extension should search for AOCE templates. A file type template can contain the resources listed in Table 5-13.

Table 5-13 Resources in file type templates

Resource type	Offset of resource ID from signature resource ID	Purpose of resource
`'detx'`	0	Identifies template as file type and provides a base resource ID. Required for all file type templates.
`'rstr'`	`kDETTemplateName`	Name of template. Required for all file type templates.

File Type Template Signature Resource

The file type template signature resource provides a base resource ID from which the other resource IDs in the template are offset, provides the file type template version number of the template, and lists the file types that the CE is to search for templates.

'detx' Resource

Use a resource of type `'detx'` for the file type template signature resource.

```
type 'detx' {
   longInt = kDETFileTypeVersion;      /* template format version */
   integer = $$CountOf(ItemArray);     /* count */
   array ItemArray {
      longInt;                         /* type of additional file */
   };
};
```

File Type Template Resources

A file type template contains a resource that names the template. The file type template name resource is described in “Template Names” on page 5-75.